HNCO

0.13

Generated by Doxygen 1.8.13

Contents

1	Nan	nespace	Index		1
	1.1	Names	space List		1
2	Hier	archica	l Index		3
	2.1	Class I	Hierarchy		3
3	Clas	ss Index			7
	3.1	Class I	List		7
4	Nam	nespace	Docume	ntation	13
	4.1	hnco N	lamespace	e Reference	13
		4.1.1		Description	
		4.1.2		Documentation	
		1.1.2	4.1.2.1	bit t	
			4.1.2.1		
			4.1.2.2	sparse_bit_matrix_t	18
			4.1.2.3	sparse_bit_vector_t	18
			4.1.2.4	transvection_sequence_t	19
		4.1.3	Function	Documentation	19
			4.1.3.1	bm_add_columns()	19
			4.1.3.2	bm_add_rows()	19
			4.1.3.3	bm_identity() [1/2]	20
			4.1.3.4	bm_identity() [2/2]	20
			4.1.3.5	bm_invert()	20
			4.1.3.6	bm_multiply()	21
			4.1.3.7	bm_rank()	21

ii CONTENTS

	4.1.3.8	bm_row_echelon_form()	22
	4.1.3.9	bm_solve()	22
	4.1.3.10	bm_solve_upper_triangular()	23
	4.1.3.11	bv_from_vector_bool()	23
	4.1.3.12	bv_to_vector_bool()	24
	4.1.3.13	is_in_range() [1/2]	24
	4.1.3.14	is_in_range() [2/2]	24
	4.1.3.15	perm_identity()	25
	4.1.3.16	perm_random()	25
	4.1.3.17	sbm_multiply()	25
	4.1.3.18	ts_is_valid() [1/2]	25
	4.1.3.19	ts_is_valid() [2/2]	26
	4.1.3.20	ts_multiply() [1/2]	26
	4.1.3.21	ts_multiply() [2/2]	27
	4.1.3.22	ts_random()	27
	4.1.3.23	ts_random_commuting()	28
	4.1.3.24	ts_random_disjoint()	28
	4.1.3.25	ts_random_non_commuting()	29
	4.1.3.26	ts_random_unique_destination()	29
	4.1.3.27	ts_random_unique_source()	30
4.2	hnco::algorithm 1	Namespace Reference	30
	4.2.1 Detailed	Description	32
4.3	hnco::algorithm::	bm_pbil Namespace Reference	33
	4.3.1 Detailed	Description	33
4.4	hnco::algorithm::	thea Namespace Reference	33
	4.4.1 Detailed	Description	33
4.5	hnco::exception l	Namespace Reference	33
	4.5.1 Detailed	Description	34
4.6	hnco::function Na	amespace Reference	34
	4.6.1 Detailed	Description	36
4.7	hnco::function::re	eal Namespace Reference	36
	4.7.1 Detailed	Description	37
4.8	hnco::neighborho	ood Namespace Reference	37
	4.8.1 Detailed	Description	37
4.9	hnco::random Na	amespace Reference	38
	4.9.1 Detailed	Description	38

CONTENTS

5	Clas	s Docu	mentation	39
	5.1	Abstra	ctLabs Class Reference	39
		5.1.1	Detailed Description	40
	5.2	Abstra	ctMaxSat Class Reference	40
		5.2.1	Detailed Description	41
		5.2.2	Member Function Documentation	41
			5.2.2.1 load()	41
		5.2.3	Member Data Documentation	41
			5.2.3.1 _expression	41
	5.3	Additiv	eGaussianNoise Class Reference	42
		5.3.1	Detailed Description	43
	5.4	Affine	Map Class Reference	43
		5.4.1	Detailed Description	44
		5.4.2	Member Function Documentation	44
			5.4.2.1 is_surjective()	44
			5.4.2.2 random()	44
	5.5	Algorit	hm Class Reference	45
		5.5.1	Detailed Description	47
		5.5.2	Member Function Documentation	47
			5.5.2.1 set_solution()	47
			5.5.2.2 update_solution()	48
		5.5.3	Member Data Documentation	48
			5.5.3.1 _functions	48
	5.6	Bernou	ulliProcess Class Reference	48
		5.6.1	Detailed Description	49
		5.6.2	Constructor & Destructor Documentation	49
			5.6.2.1 BernoulliProcess() [1/2]	49
			5.6.2.2 BernoulliProcess() [2/2]	50
		5.6.3	Member Function Documentation	50
			5.6.3.1 set_allow_stay()	50

iv CONTENTS

		5.6.3.2 set_probability()	. 50
5.7	Biased	Crossover Class Reference	. 51
	5.7.1	Detailed Description	. 51
	5.7.2	Member Function Documentation	. 51
		5.7.2.1 breed()	. 51
5.8	BitHero	ding Class Reference	. 52
	5.8.1	Detailed Description	. 53
	5.8.2	Member Enumeration Documentation	. 53
		5.8.2.1 anonymous enum	. 53
5.9	BitMon	nent Struct Reference	. 54
	5.9.1	Detailed Description	. 55
5.10	BmPbil	l Class Reference	. 55
	5.10.1	Detailed Description	. 57
	5.10.2	Member Enumeration Documentation	. 57
		5.10.2.1 anonymous enum	. 57
		5.10.2.2 anonymous enum	. 58
		5.10.2.3 anonymous enum	. 58
	5.10.3	Member Function Documentation	. 58
		5.10.3.1 set_selection_size()	. 58
5.11	Cache	Class Reference	. 59
	5.11.1	Detailed Description	. 60
	5.11.2	Constructor & Destructor Documentation	. 60
		5.11.2.1 Cache()	. 60
	5.11.3	Member Function Documentation	. 60
		5.11.3.1 provides_incremental_evaluation()	. 61
5.12	CallCo	unter Class Reference	. 61
	5.12.1	Detailed Description	. 62
5.13	Compa	actGa Class Reference	. 62
	5.13.1	Detailed Description	. 64
5.14	Comple	eteSearch Class Reference	. 64

CONTENTS

	5.14.1	Detailed Description	65
5.15	Crosso	over Class Reference	65
	5.15.1	Detailed Description	65
	5.15.2	Member Function Documentation	66
		5.15.2.1 breed()	66
5.16	Decept	tiveJump Class Reference	66
	5.16.1	Detailed Description	67
	5.16.2	Member Function Documentation	67
		5.16.2.1 get_maximum()	67
		5.16.2.2 has_known_maximum()	68
5.17	Dyadic	RealRepresentation Class Reference	68
	5.17.1	Detailed Description	69
	5.17.2	Constructor & Destructor Documentation	69
		5.17.2.1 DyadicRealRepresentation()	69
5.18	EqualF	Products Class Reference	70
	5.18.1	Detailed Description	71
	5.18.2	Member Function Documentation	71
		5.18.2.1 generate()	71
		5.18.2.2 random()	71
5.19	Error C	Class Reference	73
	5.19.1	Detailed Description	74
5.20	Progre	ssTracker::Event Struct Reference	74
	5.20.1	Detailed Description	74
5.21	Except	tion Class Reference	74
	5.21.1	Detailed Description	75
5.22	Factori	zation Class Reference	75
	5.22.1	Detailed Description	76
	5.22.2	Constructor & Destructor Documentation	76
		5.22.2.1 Factorization()	76
	5.22.3	Member Function Documentation	77

vi

		5.22.3.1 load()	77
5.23	FirstAs	centHillClimbing Class Reference	77
	5.23.1	Detailed Description	78
5.24	FourPe	eaks Class Reference	79
	5.24.1	Detailed Description	80
	5.24.2	Member Function Documentation	80
		5.24.2.1 get_maximum()	80
		5.24.2.2 has_known_maximum()	81
5.25	Function	on Class Reference	81
	5.25.1	Detailed Description	83
	5.25.2	Member Function Documentation	83
		5.25.2.1 compute_walsh_transform()	83
		5.25.2.2 get_maximum()	84
		5.25.2.3 incremental_eval()	84
		5.25.2.4 provides_incremental_evaluation()	85
		5.25.2.5 safe_eval()	85
5.26	Function	onController Class Reference	86
	5.26.1	Detailed Description	87
	5.26.2	Member Function Documentation	87
		5.26.2.1 provides_incremental_evaluation()	87
5.27	Function	onDecorator Class Reference	87
	5.27.1	Detailed Description	88
5.28	Function	onMapComposition Class Reference	88
	5.28.1	Detailed Description	89
	5.28.2	Constructor & Destructor Documentation	89
		5.28.2.1 FunctionMapComposition()	89
	5.28.3	Member Function Documentation	90
		5.28.3.1 get_maximum()	90
		5.28.3.2 has_known_maximum()	90
5.29	Function	onModifier Class Reference	91

CONTENTS vii

	5.29.1	Detailed Description	91
5.30	Function	nPlugin Class Reference	92
	5.30.1	Detailed Description	93
	5.30.2	Constructor & Destructor Documentation	93
		5.30.2.1 FunctionPlugin()	93
5.31	Genetic	cAlgorithm Class Reference	93
	5.31.1	Detailed Description	95
	5.31.2	Constructor & Destructor Documentation	95
		5.31.2.1 GeneticAlgorithm()	95
	5.31.3	Member Function Documentation	96
		5.31.3.1 set_allow_stay()	96
5.32	Hammi	ngBall Class Reference	96
	5.32.1	Detailed Description	97
	5.32.2	Constructor & Destructor Documentation	97
		5.32.2.1 HammingBall()	97
5.33	Hammi	ngSphere Class Reference	98
	5.33.1	Detailed Description	99
	5.33.2	Constructor & Destructor Documentation	99
		5.33.2.1 HammingSphere()	99
5.34	Hammi	ngSphereIterator Class Reference	99
	5.34.1	Detailed Description	100
	5.34.2	Constructor & Destructor Documentation	101
		5.34.2.1 HammingSphereIterator()	101
5.35	Hboa C	Class Reference	101
	5.35.1	Detailed Description	102
5.36	Hea< I	Moment, Herding > Class Template Reference	102
	5.36.1	Detailed Description	105
	5.36.2	Member Enumeration Documentation	105
		5.36.2.1 anonymous enum	105
	5.36.3	Constructor & Destructor Documentation	105

viii CONTENTS

		5.36.3.1 Hea()
	5.36.4	Member Function Documentation
		5.36.4.1 set_reset_period()
		5.36.4.2 set_selection_size()
5.37	Hiff Cla	ass Reference
	5.37.1	Detailed Description
	5.37.2	Member Function Documentation
		5.37.2.1 get_maximum()
		5.37.2.2 has_known_maximum()
5.38	HncoE	valuator Class Reference
	5.38.1	Detailed Description
5.39	Hypero	cubelterator Class Reference
	5.39.1	Detailed Description
5.40	Injectio	on Class Reference
	5.40.1	Detailed Description
	5.40.2	Constructor & Destructor Documentation
		5.40.2.1 Injection()
5.41	Iterative	eAlgorithm Class Reference
	5.41.1	Detailed Description
	5.41.2	Constructor & Destructor Documentation
		5.41.2.1 IterativeAlgorithm()
	5.41.3	Member Function Documentation
		5.41.3.1 maximize()
		5.41.3.2 set_num_iterations()
5.42	Iterator	Class Reference
	5.42.1	Detailed Description
5.43	Jump (Class Reference
	5.43.1	Detailed Description
	5.43.2	Member Function Documentation
		5.43.2.1 get_maximum()

CONTENTS

		5.43.2.2 has_known_maximum()	17
5.44	Labs C	ass Reference	18
	5.44.1	Detailed Description	18
5.45	LabsM	eritFactor Class Reference	19
	5.45.1	Detailed Description	19
5.46	LastEv	aluation Class Reference	20
	5.46.1	Detailed Description	20
5.47	Leading	gOnes Class Reference	20
	5.47.1	Detailed Description	21
	5.47.2	Member Function Documentation	21
		5.47.2.1 get_maximum()	21
		5.47.2.2 has_known_maximum()	22
5.48	LinearF	runction Class Reference	22
	5.48.1	Detailed Description	23
	5.48.2	Member Function Documentation	24
		5.48.2.1 generate()	24
		5.48.2.2 has_known_maximum()	24
		5.48.2.3 provides_incremental_evaluation()	24
		5.48.2.4 random()	24
5.49	Linear	Map Class Reference 1	25
	5.49.1	Detailed Description	26
	5.49.2	Member Function Documentation	26
		5.49.2.1 is_surjective()	26
		5.49.2.2 random()	26
5.50	LocalM	aximum Class Reference	27
	5.50.1	Detailed Description	28
5.51	LogCo	ntext Class Reference	28
	5.51.1	Detailed Description	28
5.52	LongPa	th Class Reference	29
	5.52.1	Detailed Description	30

CONTENTS

	5.52.2	Member Function Documentation	30
		5.52.2.1 get_maximum()	30
		5.52.2.2 has_known_maximum()	31
5.53	Ltga Cl	lass Reference	31
	5.53.1	Detailed Description	32
5.54	Map Cl	lass Reference	32
	5.54.1	Detailed Description	33
	5.54.2	Member Function Documentation	34
		5.54.2.1 is_surjective()	34
5.55	MapCo	emposition Class Reference	34
	5.55.1	Detailed Description	35
	5.55.2	Constructor & Destructor Documentation	35
		5.55.2.1 MapComposition()	35
	5.55.3	Member Function Documentation	36
		5.55.3.1 is_surjective()	36
5.56	Maxim	umReached Class Reference	36
	5.56.1	Detailed Description	37
5.57	MaxNa	e3Sat Class Reference	37
	5.57.1	Detailed Description	38
	5.57.2	Member Function Documentation	38
		5.57.2.1 load()	38
5.58	MaxSa	t Class Reference	38
	5.58.1	Detailed Description	39
	5.58.2	Member Function Documentation	39
		5.58.2.1 random() [1/2]	39
		5.58.2.2 random() [2/2]	40
5.59	Mimic (Class Reference	40
	5.59.1	Detailed Description	42
5.60	Mmas	Class Reference	43
	5.60.1	Detailed Description	44

CONTENTS xi

5.61	Model	Class Reference	44
	5.61.1	Detailed Description	45
5.62	ModelF	Parameters Class Reference	45
	5.62.1	Detailed Description	46
5.63	MuCon	nmaLambdaEa Class Reference	46
	5.63.1	Detailed Description	47
	5.63.2	Constructor & Destructor Documentation	47
		5.63.2.1 MuCommaLambdaEa()	47
	5.63.3	Member Function Documentation	48
		5.63.3.1 set_allow_stay()	48
5.64	MultiBi	tFlip Class Reference	48
	5.64.1	Detailed Description	49
	5.64.2	Constructor & Destructor Documentation	49
		5.64.2.1 MultiBitFlip()	49
	5.64.3	Member Function Documentation	49
		5.64.3.1 bernoulli_trials()	49
		5.64.3.2 reservoir_sampling()	50
5.65	MuPlus	sLambdaEa Class Reference	50
	5.65.1	Detailed Description	51
	5.65.2	Constructor & Destructor Documentation	52
		5.65.2.1 MuPlusLambdaEa()	52
	5.65.3	Member Function Documentation	52
		5.65.3.1 set_allow_stay()	52
5.66	Neares	stNeighborlsingModel1 Class Reference	53
	5.66.1	Detailed Description	54
	5.66.2	Member Function Documentation	55
		5.66.2.1 eval()	55
		5.66.2.2 generate()	55
		5.66.2.3 provides_incremental_evaluation()	55
		5.66.2.4 random()	56

xii CONTENTS

5.67	Neares	stNeighborIsingModel2 Class Reference	6
	5.67.1	Detailed Description	58
	5.67.2	Member Function Documentation	58
		5.67.2.1 eval()	58
		5.67.2.2 generate()	58
		5.67.2.3 provides_incremental_evaluation()	59
		5.67.2.4 random()	59
5.68	Needle	e Class Reference	30
	5.68.1	Detailed Description	30
	5.68.2	Member Function Documentation	31
		5.68.2.1 get_maximum()	31
		5.68.2.2 has_known_maximum()	31
5.69	Negatio	on Class Reference	32
	5.69.1	Detailed Description	3
	5.69.2	Member Function Documentation	3
		5.69.2.1 provides_incremental_evaluation()	3
5.70	Neighb	porhood Class Reference	34
	5.70.1	Detailed Description	35
	5.70.2	Constructor & Destructor Documentation	35
		5.70.2.1 Neighborhood()	35
	5.70.3	Member Function Documentation	36
		5.70.3.1 map()	36
		5.70.3.2 mutate()	36
5.71	Neighb	porhoodIterator Class Reference	37
	5.71.1	Detailed Description	37
	5.71.2	Constructor & Destructor Documentation	37
		5.71.2.1 NeighborhoodIterator()	37
5.72	NkLand	dscape Class Reference	8
	5.72.1	Detailed Description	39
	5.72.2	Member Function Documentation	39

CONTENTS xiii

		5.72.2.1	generate()	 	 	169
		5.72.2.2	random()	 	 	170
		5.72.2.3	random_structure()	 	 	170
5.73	NpsPbi	il Class Re	erence	 	 	170
	5.73.1	Detailed I	escription	 	 	172
5.74	OnBud	getFunctio	Class Reference	 	 	172
	5.74.1	Detailed I	escription	 	 	173
	5.74.2	Member F	unction Documentation	 	 	173
		5.74.2.1	eval()	 	 	173
		5.74.2.2	incremental_eval()	 	 	174
		5.74.2.3	update()	 	 	174
5.75	OneMa	ax Class Re	ference	 	 	174
	5.75.1	Detailed I	escription	 	 	175
	5.75.2	Member F	unction Documentation	 	 	176
		5.75.2.1	get_maximum()	 	 	176
		5.75.2.2	has_known_maximum()	 	 	176
		5.75.2.3	provides_incremental_evaluation()	 	 	176
5.76	OnePlu	usLambda(ommaLambdaGa Class Reference	 	 	177
	5.76.1	Detailed I	escription	 	 	178
	5.76.2	Construct	or & Destructor Documentation	 	 	178
		5.76.2.1	OnePlusLambdaCommaLambdaGa()	 	 	178
5.77	OnePlu	usOneEa C	ass Reference	 	 	179
	5.77.1	Detailed I	escription	 	 	180
	5.77.2	Construct	or & Destructor Documentation	 	 	180
		5.77.2.1	OnePlusOneEa()	 	 	180
	5.77.3	Member F	unction Documentation	 	 	181
		5.77.3.1	set_allow_stay()	 	 	181
		5.77.3.2	set_num_iterations()	 	 	181
5.78	Parame	eterLessPo	pulationPyramid Class Reference	 	 	181
	5.78.1	Detailed I	escription	 	 	182

xiv CONTENTS

5.79	Parsed	Modifier Cl	ass Referer	nce			 	 	 	 	 183
	5.79.1	Detailed D	Description				 	 	 	 	 184
	5.79.2	Constructo	or & Destru	ctor Docur	mentatior	n	 	 	 	 	 184
		5.79.2.1	ParsedMod	lifier()			 	 	 	 	 184
5.80	Parsed	RealMultiva	ariateFuncti	on Class F	Referenc	е	 	 	 	 	 184
	5.80.1	Detailed D	Description				 	 	 	 	 185
	5.80.2	Constructo	or & Destru	ctor Docur	mentatior	1	 	 	 	 	 185
		5.80.2.1	ParsedRea	lMultivaria	ıteFunctio	on() .	 	 	 	 	 185
5.81	Pbil Cla	ass Referer	nce				 	 	 	 	 186
	5.81.1	Detailed D	Description				 	 	 	 	 187
5.82	Permut	ation Class	Reference				 	 	 	 	 187
	5.82.1	Detailed D	Description				 	 	 	 	 188
	5.82.2	Member F	unction Do	cumentatio	on		 	 	 	 	 188
		5.82.2.1	is_surjectiv	e()			 	 	 	 	 189
5.83	Plateau	ı Class Ref	erence				 	 	 	 	 189
	5.83.1	Detailed D	Description				 	 	 	 	 190
	5.83.2	Member F	unction Do	cumentatio	on		 	 	 	 	 190
		5.83.2.1	get_maxim	um()			 	 	 	 	 190
		5.83.2.2	has_known	ı_maximur	m()		 	 	 	 	 190
5.84	PointVa	alueExcepti	on Class R	eference			 	 	 	 	 191
	5.84.1	Detailed D	Description				 	 	 	 	 191
5.85	Populat	tion Class F	Reference .				 	 	 	 	 192
	5.85.1	Detailed D	Description				 	 	 	 	 193
	5.85.2	Member F	unction Do	cumentatio	on		 	 	 	 	 194
		5.85.2.1	comma_se	lection() [1/2]		 	 	 	 	 194
		5.85.2.2	comma_se	lection() [2/2]		 	 	 	 	 194
		5.85.2.3	get_best_b	v() [1/4]			 	 	 	 	 195
		5.85.2.4	get_best_b	v() [2/4]			 	 	 	 	 195
		5.85.2.5	get_best_b	v() [3/4]			 	 	 	 	 195
		5.85.2.6	get_best_b	v() [4/4]			 	 	 	 	 195

CONTENTS xv

		5.85.2.7 get_best_value() [1/2]	96
		5.85.2.8 get_best_value() [2/2]	96
		5.85.2.9 get_worst_bv() [1/2]	96
		5.85.2.10 get_worst_bv() [2/2]	97
		5.85.2.11 plus_selection() [1/2]	97
		5.85.2.12 plus_selection() [2/2]	98
	5.85.3	Member Data Documentation	98
		5.85.3.1 _compare_index_value	98
		5.85.3.2 _lookup	98
5.86	PriorNo	pise Class Reference	99
	5.86.1	Detailed Description	200
	5.86.2	Member Function Documentation	200
		5.86.2.1 get_maximum()	200
		5.86.2.2 has_known_maximum()	200
		5.86.2.3 provides_incremental_evaluation()	201
5.87	Progres	ssTracker Class Reference	201
	5.87.1	Detailed Description	203
	5.87.2	Member Function Documentation	203
		5.87.2.1 get_last_improvement()	203
5.88	Progres	ssTrackerContext Class Reference	203
	5.88.1	Detailed Description	204
5.89	Project	tion Class Reference	204
	5.89.1	Detailed Description	205
	5.89.2	Constructor & Destructor Documentation	205
		5.89.2.1 Projection()	205
	5.89.3	Member Function Documentation	206
		5.89.3.1 is_surjective()	206
5.90	PvAlgo	orithm Class Reference	206
	5.90.1	Detailed Description	208
5.91	Qubo C	Class Reference	208

xvi CONTENTS

	5.91.1	Detailed Description	209
	5.91.2	Member Function Documentation	209
		5.91.2.1 load()	209
	5.91.3	Member Data Documentation	210
		5.91.3.1 _q	210
5.92	Rando	m Struct Reference	210
	5.92.1	Detailed Description	211
5.93	Randoi	mLocalSearch Class Reference	211
	5.93.1	Detailed Description	212
	5.93.2	Member Function Documentation	212
		5.93.2.1 set_patience()	212
5.94	Randoi	mSearch Class Reference	213
	5.94.1	Detailed Description	214
5.95	Randoi	mWalk Class Reference	214
	5.95.1	Detailed Description	215
5.96	RealMu	ultivariateFunction Class Reference	215
	5.96.1	Detailed Description	216
5.97	RealMu	ultivariateFunctionAdapter Class Reference	216
	5.97.1	Detailed Description	217
	5.97.2	Constructor & Destructor Documentation	217
		5.97.2.1 RealMultivariateFunctionAdapter()	217
5.98	RealRe	epresentation Class Reference	218
	5.98.1	Detailed Description	218
5.99	Restar	t Class Reference	219
	5.99.1	Detailed Description	220
5.100	ORidge (Class Reference	220
	5.100.1	Detailed Description	221
	5.100.2	2 Member Function Documentation	221
		5.100.2.1 get_maximum()	221
		5.100.2.2 has_known_maximum()	221

CONTENTS xvii

5.101 Simulated Annealing Class Reference
5.101.1 Detailed Description
5.101.2 Member Function Documentation
5.101.2.1 init_beta()
5.102SingleBitFlip Class Reference
5.102.1 Detailed Description
5.103SingleBitFlipIterator Class Reference
5.103.1 Detailed Description
5.103.2 Constructor & Destructor Documentation
5.103.2.1 SingleBitFlipIterator()
5.104SinusSummationCancellation Class Reference
5.104.1 Detailed Description
5.105SixPeaks Class Reference
5.105.1 Detailed Description
5.105.2 Member Function Documentation
5.105.2.1 get_maximum()
5.105.2.2 has_known_maximum()
5.106SpinHerding Class Reference
5.106.1 Detailed Description
5.106.2 Member Enumeration Documentation
5.106.2.1 anonymous enum
5.106.3 Constructor & Destructor Documentation
5.106.3.1 SpinHerding()
5.106.4 Member Function Documentation
5.106.4.1 q_variation()
5.107SpinMoment Struct Reference
5.107.1 Detailed Description
5.107.2 Member Data Documentation
5.107.2.1 _second
5.108SteepestAscentHillClimbing Class Reference

xviii CONTENTS

5.108.1 Detailed Description
5.109StopOnMaximum Class Reference
5.109.1 Detailed Description
5.109.2 Constructor & Destructor Documentation
5.109.2.1 StopOnMaximum()
5.109.3 Member Function Documentation
5.109.3.1 eval()
5.109.3.2 incremental_eval()
5.109.3.3 update()
5.110StopOnTarget Class Reference
5.110.1 Detailed Description
5.110.2 Constructor & Destructor Documentation
5.110.2.1 StopOnTarget()
5.110.3 Member Function Documentation
5.110.3.1 eval()
5.110.3.2 incremental_eval()
5.110.3.3 update()
5.111 Stop Watch Class Reference
5.111.1 Detailed Description
5.112SummationCancellation Class Reference
5.112.1 Detailed Description
5.112.2 Constructor & Destructor Documentation
5.112.2.1 SummationCancellation()
5.112.3 Member Function Documentation
5.112.3.1 has_known_maximum()
5.113TargetReached Class Reference
5.113.1 Detailed Description
5.114TournamentSelection Class Reference
5.114.1 Detailed Description
5.114.2 Member Function Documentation

CONTENTS xix

5.114.2.1 select()
5.115Translation Class Reference
5.115.1 Detailed Description
5.115.2 Member Function Documentation
5.115.2.1 is_surjective()
5.116Transvection Struct Reference
5.116.1 Detailed Description
5.116.2 Member Function Documentation
5.116.2.1 is_valid()
5.116.2.2 multiply() [1/2]
5.116.2.3 multiply() [2/2]
5.116.2.4 multiply_right()
5.116.2.5 random()
5.116.2.6 random_non_commuting()
5.117Trap Class Reference
5.117.1 Detailed Description
5.117.2 Constructor & Destructor Documentation
5.117.2.1 Trap()
5.117.3 Member Function Documentation
5.117.3.1 get_maximum()
5.117.3.2 has_known_maximum()
5.118TsAffineMap Class Reference
5.118.1 Detailed Description
5.118.2 Member Enumeration Documentation
5.118.2.1 SamplingMode
5.118.3 Member Function Documentation
5.118.3.1 is_surjective()
5.118.3.2 random()
5.119Umda Class Reference
5.119.1 Detailed Description

CONTENTS

5.120UniformCrossover Class Reference	259
5.120.1 Detailed Description	259
5.120.2 Member Function Documentation	260
5.120.2.1 breed()	260
5.121WalshExpansion Class Reference	260
5.121.1 Detailed Description	261
5.121.2 Member Function Documentation	262
5.121.2.1 generate()	262
5.121.2.2 random()	262
5.122WalshExpansion1 Class Reference	262
5.122.1 Detailed Description	264
5.122.2 Member Function Documentation	264
5.122.2.1 generate()	264
5.122.2.2 has_known_maximum()	264
5.122.2.3 provides_incremental_evaluation()	265
5.122.2.4 random()	265
5.123WalshExpansion2 Class Reference	265
5.123.1 Detailed Description	267
5.123.2 Member Function Documentation	267
5.123.2.1 generate()	267
5.123.2.2 generate_ising1_long_range()	267
5.123.2.3 generate_ising1_long_range_periodic()	268
5.123.2.4 random()	269
5.123.3 Member Data Documentation	269
5.123.3.1 _quadratic	269
5.124Function::WalshTransformTerm Struct Reference	269
5.124.1 Detailed Description	270
5.124.2 Member Data Documentation	270
5.124.2.1 feature	270
Index	271

Chapter 1

Namespace Index

1.1 Namespace List

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

TITICO	
Top-level HNCO namespace	13
hnco::algorithm	
Algorithms	30
hnco::algorithm::bm_pbil	
Boltzmann machine PBIL	33
hnco::algorithm::hea	
Herding evolutionary algorithm	33
hnco::exception	
Exceptions	33
hnco::function	
Real black box functions defined on bit vectors	34
hnco::function::real	
Real multivariate functions	36
hnco::neighborhood	
Neighborhoods for local search	37
hnco::random	
Pseudo random numbers	38

2 Namespace Index

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Algorithm
CompleteSearch
Hboa
Ltga
ParameterLessPopulationPyramid
IterativeAlgorithm
BmPbil
Mimic
FirstAscentHillClimbing
GeneticAlgorithm
Hea < Moment, Herding >
MuCommaLambdaEa
MuPlusLambdaEa
OnePlusLambdaCommaLambdaGa
PvAlgorithm
CompactGa
Mmas
NpsPbil
Pbil
Umda
RandomLocalSearch
RandomSearch
RandomWalk
Restart
SimulatedAnnealing
SteepestAscentHillClimbing
OnePlusOneEa
BitHerding
BitMoment
Crossover
BiasedCrossover
UniformCrossover
Evaluator
HncoEvaluator

4 Hierarchical Index

ProgressTracker::Event	
Exception	 74
Error	 . 73
LastEvaluation	 120
PointValueException	 191
LocalMaximum	 127
MaximumReached	 136
TargetReached	 244
Function	 81
AbstractLabs	 39
Labs	
LabsMeritFactor	
AbstractMaxSat	
MaxNae3Sat	
MaxSat	
DeceptiveJump	
EqualProducts	
Factorization	
FourPeaks	
FunctionDecorator	
FunctionController	
Cache	
CallCounter	
OnBudgetFunction	_
ProgressTracker	
StopOnMaximum	
•	
StopOnTarget	
AdditiveGaussianNoise	
FunctionMapComposition	
Negation	
ParsedModifier	
PriorNoise	
FunctionPlugin	
Hiff	
Jump	
LeadingOnes	
LinearFunction	
LongPath	
NearestNeighborIsingModel2	
Needle	
NkLandscape	
OneMax	
Plateau	
Qubo	
RealMultivariateFunctionAdapter	
Ridge	
SixPeaks	
SummationCancellation	
SinusSummationCancellation	
Trap	
WalshExpansion	
WalshExpansion1	
WalshExpansion2	
Iterator	
Hypercubelterator	 . 109

2.1 Class Hierarchy 5

NeighborhoodIterator	167
HammingSphereIterator	99
SingleBitFlipIterator	225
LogContext	128
ProgressTrackerContext	203
Map	132
AffineMap	43
Injection	
LinearMap	
MapComposition	134
Permutation	187
Projection	204
Translation	
TsAffineMap	255
Model	144
ModelParameters	145
Neighborhood	164
MultiBitFlip	148
BernoulliProcess	48
HammingBall	96
HammingSphere	98
SingleBitFlip	224
Population	192
TournamentSelection	245
Random	
RealMultivariateFunction	
ParsedRealMultivariateFunction	184
RealRepresentation	
DyadicRealRepresentation	
SpinHerding	
SpinMoment	
StopWatch	
Transvection	
Function::WalshTransformTerm	

6 Hierarchical Index

Chapter 3

Class Index

3.1 Class List

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

AbstractLabs	
Abstract class for low autocorrelation binary sequences	39
AbstractMaxSat	
Abstract class for MaxSat-like functions	40
AdditiveGaussianNoise	
Additive Gaussian Noise	42
AffineMap	
Affine map	43
Algorithm	
Abstract search algorithm	45
BernoulliProcess	
Bernoulli process	48
BiasedCrossover	
Biased crossover	51
BitHerding	
Herding with bit features	52
BitMoment	
Moment for bit features	54
BmPbil	
Boltzmann machine PBIL	55
Cache	
Cache	59
CallCounter	
Call counter	61
CompactGa	00
Compact genetic algorithm	62
CompleteSearch	C 4
Complete search	64
Crossover	C.F.
Crossover	65
Deceptive jump	66
DyadicRealRepresentation	66
Dyadic real representation	68
EqualProducts	UO
Equal products	70

8 Class Index

Error	_	
	Error	73
1	Event	74
Exception		74
Factorizat		75
	ntHillClimbing	75
		77
FourPeak		
Function	Four Peaks	79
	Function	81
FunctionC		
		86
FunctionD		87
	ApComposition	07
		88
FunctionN		
		91
FunctionP	•	92
GeneticAl	gorithm	
Hamming	o	93
•		96
Hamming	Sphere	
	0 1	98
•	SphereIterator Hamming sphere neighborhood iterator	99
Hboa		•
		01
	ment, Herding >	00
Hiff	Herding evolutionary algorithm	02
	Hierarchical if and only if	07
HncoEval		
	Evaluator for HNCO functions	80
Hypercub	etterator Hypercube iterator	09
Injection		00
	Injection	10
IterativeAl		
Iterator	Iterative search	12
	Iterator over bit vectors	15
Jump		
	Jump	16
Labs	Low autocorrelation binary sequences	12
LabsMerit		10
	Low autocorrelation binary sequences merit factor	19
LastEvalu		_
LeadingO	Last evaluation	20
	Leading ones	20

3.1 Class List

LinearFur	nction	
		122
LinearMa	p Linear map	125
LocalMax	imum Local maximum	127
LogConte		
LongPath		
Ltga	Long path	129
Мар		131
MapComp	Map	132
Maximum	Map composition	134
	Maximum reached	136
MaxNae3		137
MaxSat	MAX-SAT	138
Mimic	Mutual information maximizing input clustering	140
Mmas		
Model		
ModelPar	ameters	144
MuComm	Parameters of a Boltzmann machine	
MultiBitFli	(mu, lambda) EA	146
	Multi bit flip	148
	(mu+lambda) EA	150
	eighborlsingModel1 Nearest neighbor Ising model in one dimension	153
	eighborlsingModel2 Nearest neighbor Ising model in two dimensions	156
Needle	Needle in a haystack	160
Negation		162
Neighborl	nood	
Neighborl	noodIterator	164
NkLandso		
NpsPbil	NK landscape	168
•	Population-based incremental learning with negative and positive selection	170
		172
	OneMax	174
	.ambdaCommaLambdaGa (1+(lambda, lambda)) genetic algorithm	177

10 Class Index

OnePlusOneEa		
(1+1) EA		179
ParameterLessPopulationPyramid		404
Parameter-less Population Pyramid		181
Parsed modifier		183
ParsedRealMultivariateFunction		
Parsed real multivariate function		184
Pbil		
Population-based incremental learning		186
Permutation Permut		187
Plateau	• •	107
Plateau		189
PointValueException		
Point-value exception		191
Population		400
Population		192
Prior noise		199
ProgressTracker		
ProgressTracker		201
ProgressTrackerContext		
Log context for ProgressTracker	٠.	203
Projection		204
PvAlgorithm		204
Probability vector algorithm		206
Qubo		
Quadratic unconstrained binary optimization		208
Random Random numbers		210
RandomLocalSearch	• •	210
Random local search		211
RandomSearch		
Random search		213
RandomWalk		
Random walk	• •	214
Real multivariate function		215
RealMultivariateFunctionAdapter		
Real multivariate function adapter		216
RealRepresentation		
Real representation	٠.	218
Restart Restart		219
Ridge		213
Ridge		220
SimulatedAnnealing		
Simulated annealing		222
SingleBitFlip One bit neighborhood		004
SingleBitFlipIterator		224
Single bit flip neighborhood iterator		225
SinusSummationCancellation		
Summation cancellation with sinus		226
SixPeaks		
Six Peaks		227

3.1 Class List

SpinHerding	
Herding with spin variables	229
SpinMoment	
Moment for spin variables	232
SteepestAscentHillClimbing	
Steepest ascent hill climbing	234
StopOnMaximum	
Stop on maximum	235
StopOnTarget	
Stop on target	238
StopWatch	
Stop watch	241
SummationCancellation	
Summation cancellation	242
TargetReached	
Target reached	244
TournamentSelection	
Population with tournament selection	245
Translation	
Translation	247
Transvection	
Transvection	248
Trap	
Trap	252
TsAffineMap	
Transvection sequence affine map	255
Umda	
Univariate marginal distribution algorithm	257
UniformCrossover	
Uniform crossover	259
WalshExpansion	
Walsh expansion	260
WalshExpansion1	
Walsh expansion of degree 1	262
WalshExpansion2	
Walsh expansion of degree 2	265
Function::WalshTransformTerm	
Walsh transform term	269

12 Class Index

Chapter 4

Namespace Documentation

4.1 hnco Namespace Reference

top-level HNCO namespace

Namespaces

algorithm

Algorithms.

• exception

Exceptions.

function

Real black box functions defined on bit vectors.

· neighborhood

Neighborhoods for local search.

• random

Pseudo random numbers.

Classes

class AffineMap

Affine map.

· class Hypercubelterator

Hypercube iterator.

class Injection

Injection.

· class Iterator

Iterator over bit vectors

class LinearMap

Linear map.

class Map

Мар

• class MapComposition

Map composition.

class Permutation

Permutation.

class Projection

Projection.

class StopWatch

Stop watch.

class Translation

Translation.

· struct Transvection

Transvection.

class TsAffineMap

Transvection sequence affine map.

Functions

• template<class T >

T square (T x)

Generic square function.

• double logistic (double x)

Logistic function (sigmoid)

Range checking

bool is_in_range (int i, int n)

Check whether an index is in a given range.

• bool is_in_range (int i, int a, int b)

Check whether an index is in a given range.

Types and functions related to bit matrices

typedef std::vector< bit_vector_t > bit_matrix_t

Bit matrix.

void bm_display (const bit_matrix_t &M, std::ostream &stream)

Display bit matrix.

bool bm_is_valid (const bit_matrix_t &M)

Check whether a bit matrix is valid.

int bm num rows (const bit matrix t &M)

Number of rows.

int 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, int num_rows, int num_columns)

Resize a bit matrix.

void bm resize (bit matrix t &M, int 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_identity (bit_matrix_t &M, int n)

      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, int i, int j)

      Swap two rows.
• void bm_add_rows (bit_matrix_t &M, int i, int j)
      Add two rows.
• void bm_add_columns (bit_matrix_t &M, int src, int dest)
     Add two columns.
void bm_row_echelon_form (bit_matrix_t &A)
      Compute a row echelon form of a matrix.
int bm_rank (const bit_matrix_t &A)
      Compute the rank of a matrix.

    bool bm_solve (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system.

    bool bm_solve_upper_triangular (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system in upper triangular form.

    bool bm_invert (bit_matrix_t &M, bit_matrix_t &N)

      Invert a bit matrix.

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

      Multiply a bit matrix and a bit vector.

    void bm_transpose (const bit_matrix_t &M, bit_matrix_t &N)

      Transpose.
```

Types and functions related to bit

```
typedef char bit_t

Bit.
bit_t bit_flip (bit_t b)

Flip bit.
bit_t bit_random (double p)

Sample a random bit.
```

Types and functions related to bit vectors

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

Bit vector.
typedef std::pair< bit_vector_t, double > point_value_t

Type to represent point value pairs.
void bv_display (const bit_vector_t &v, std::ostream &stream)

Display bit vector.
bool bv_is_valid (const bit_vector_t &x)

Check whether the bit vector is valid.
bool bv_is_zero (const bit_vector_t &x)
```

Check whether the bit vector is zero.

int bv_hamming_weight (const bit_vector_t &x)

Hamming weight.

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

Hamming weight.

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

Hamming distance between two bit vectors.

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

Dot product.

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

Dot product.

void bv_clear (bit_vector_t &x)

Clear bit vector.

void bv_flip (bit_vector_t &x, int 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.

void bv_from_stream (bit_vector_t &x, std::istream &stream)

Read a bit vector from a stream.

Types and functions related to permutations

typedef std::vector< int > permutation_t

Permutation type

• bool perm_is_valid (const permutation_t &permutation)

Check that a vector represents a permutation.

void perm_identity (permutation_t &s)

Identity permutation.

void perm_random (permutation_t &s)

Sample a random permutation.

Types and functions related to sparse bit matrices

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

void sbm_display (const sparse_bit_matrix_t &sbm, std::ostream &stream)

Display sparse bit matrix.

• void bm_to_sbm (const bit_matrix_t &bm, sparse_bit_matrix_t &sbm)

Convert a bit matrix to a sparse bit matrix.

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

Multiply a sparse bit matrix and a bit vector.

Types and functions related to sparse bit vectors

```
    typedef std::vector< int > 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.

Types and functions related to transvections

typedef std::vector< Transvection > transvection_sequence_t

Transvection sequence.

bool transvections_commute (const Transvection &a, const Transvection &b)

Check whether two transvections commute.

• bool ts_is_valid (const transvection_sequence_t &ts)

Check validity.

bool ts_is_valid (const transvection_sequence_t &ts, int n)

Check validity.

void ts display (const transvection sequence t &ts, std::ostream &stream)

Display a transvection sequence.

void ts_random (transvection_sequence_t &ts, int n, int t)

Sample a random transvection sequence.

void ts_random_commuting (transvection_sequence_t &ts, int n, int t)

Sample a random sequence of commuting transvections.

void ts_random_unique_source (transvection_sequence_t &ts, int n, int t)

Sample a random sequence of transvections with unique source.

• void ts_random_unique_destination (transvection_sequence_t &ts, int n, int t)

Sample a random sequence of transvections with unique destination.

void ts_random_disjoint (transvection_sequence_t &ts, int n, int t)

Sample a random sequence of disjoint transvections.

• void ts random non commuting (transvection sequence t &ts, int n, int t)

Sample a random sequence of non commuting transvections.

void ts_multiply (const transvection_sequence_t &ts, bit_vector_t &x)

Multiply a vector by a transvection sequence from the left.

void ts_multiply (const transvection_sequence_t &ts, bit_matrix_t &M)

Multiply a matrix by a transvection sequence from the left.

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.

Definition at line 49 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<int> 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.

4.1.2.4 transvection_sequence_t

```
typedef std::vector<Transvection> transvection_sequence_t
```

Transvection sequence.

The general linear group of a linear space of dimension n over the finite field F_2 is the group of invertible n by n bit matrices

Any invertible bit matrix can be expressed as a finite product of transvections.

Finite transvection sequences can then represent all invertible bit matrices.

Definition at line 164 of file transvection.hh.

4.1.3 Function Documentation

4.1.3.1 bm_add_columns()

Add two columns.

Column src is added to column dest.

Parameters

М	Bit matrix
src	Source column
dest	Destination column

Warning

M is modified by the function.

Definition at line 150 of file bit-matrix.cc.

4.1.3.2 bm_add_rows()

```
int i, int j)
```

Add two rows.

Row i is added to row j.

Definition at line 141 of file bit-matrix.cc.

Set the matrix to the identity matrix.

Precondition

```
bm_is_square(M)
```

Definition at line 56 of file bit-matrix.cc.

Set the matrix to the identity matrix.

Parameters

М	Bit matrix
n	Dimension

Definition at line 67 of file bit-matrix.cc.

```
4.1.3.5 bm_invert()
```

Invert a bit matrix.

Parameters

М	input matrix
Ν	inverse matrix

Precondition

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

Returns

true if M is invertible

Warning

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

Definition at line 268 of file bit-matrix.cc.

4.1.3.6 bm_multiply()

Multiply a bit matrix and a bit vector.

The result is y = Mx.

Definition at line 312 of file bit-matrix.cc.

4.1.3.7 bm_rank()

Compute the rank of a matrix.

Precondition

A must be in row echelon form.

Definition at line 196 of file bit-matrix.cc.

4.1.3.8 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 165 of file bit-matrix.cc.

4.1.3.9 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 214 of file bit-matrix.cc.

4.1.3.10 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 247 of file bit-matrix.cc.

4.1.3.11 bv_from_vector_bool()

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

Convert a bool vector to a bit vector.

Warning

Vectors must be of the same size.

Definition at line 155 of file bit-vector.cc.

4.1.3.12 bv_to_vector_bool()

Convert a bit vector to a bool vector.

Warning

Vectors must be of the same size.

Definition at line 142 of file bit-vector.cc.

```
4.1.3.13 is_in_range() [1/2]
```

Check whether an index is in a given range.

The lower bound is implicit and is equal to 0.

Parameters

i	Index
n	Upper bound

Returns

```
true if i \ge 0 and i < n
```

Definition at line 152 of file bit-vector.hh.

4.1.3.14 is_in_range() [2/2]

Check whether an index is in a given range.

Parameters

i	Index
а	Lower bound
b	Upper bound

Returns

```
true if i \ge a and i < b
```

Definition at line 162 of file bit-vector.hh.

4.1.3.15 perm_identity()

Identity permutation.

Warning

This function does not set the size of the permutation.

Definition at line 46 of file permutation.hh.

4.1.3.16 perm_random()

Sample a random permutation.

Warning

This function does not set the size of the permutation.

Definition at line 56 of file permutation.hh.

4.1.3.17 sbm_multiply()

Multiply a sparse bit matrix and a bit vector.

The result is y = Mx.

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

```
4.1.3.18 ts_is_valid() [1/2] bool ts_is_valid (
```

const transvection_sequence_t & ts)

Check validity.

Parameters

```
ts Transvection sequence
```

Definition at line 143 of file transvection.cc.

Check validity.

Parameters

ts	Transvection sequence
n	Dimension

Definition at line 149 of file transvection.cc.

Multiply a vector by a transvection sequence from the left.

Parameters

ts	Transvection sequence
Х	Bit vector

Precondition

```
ts_is_valid(ts)
ts_is_valid(ts, x.size())
```

Warning

This function modifies the given bit vector.

Definition at line 348 of file transvection.cc.

Multiply a matrix by a transvection sequence from the left.

Parameters

ts	Transvection sequence
М	Bit matrix

Precondition

```
ts_is_valid(ts)
ts_is_valid(ts, bm_num_rows(M))
```

Warning

This function modifies the given bit vector.

Definition at line 358 of file transvection.cc.

4.1.3.22 ts_random()

Sample a random transvection sequence.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

```
n > 1
t >= 0
```

Definition at line 165 of file transvection.cc.

4.1.3.23 ts_random_commuting()

Sample a random sequence of commuting transvections.

This function ensures that all transvections in the sequence commute.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

```
n > 1
t >= 0
```

Definition at line 176 of file transvection.cc.

4.1.3.24 ts_random_disjoint()

Sample a random sequence of disjoint transvections.

Two transvections τ_{ij} and τ_{kl} are said to be disjoint if the pairs {i,j} and {k,l} are disjoint.

If 2t>n then the sequence length is set to the largest t such that 2t<=n.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

$$n > 1$$

 $t >= 0$

Definition at line 304 of file transvection.cc.

4.1.3.25 ts_random_non_commuting()

Sample a random sequence of non commuting transvections.

This function ensures that two consecutive transvections do not commute.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

```
n > 1
t >= 0
```

Definition at line 333 of file transvection.cc.

4.1.3.26 ts_random_unique_destination()

Sample a random sequence of transvections with unique destination.

A transvection sequence with unique destination is such that, for each source, there is a unique destination.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

```
n > 1
t >= 0
```

Definition at line 271 of file transvection.cc.

4.1.3.27 ts_random_unique_source()

Sample a random sequence of transvections with unique source.

A transvection sequence with unique source is such that, for each destination, there is a unique source.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

```
n > 1
t >= 0
```

Definition at line 238 of file transvection.cc.

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 LogContext

Log context.

· class Mmas

Max-min ant system.

• class MuCommaLambdaEa

(mu, lambda) EA.

• class MuPlusLambdaEa

(mu+lambda) EA.

class NpsPbil

Population-based incremental learning with negative and positive selection.

· class OnePlusLambdaCommaLambdaGa

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

• class OnePlusOneEa

(1+1) EA.

· class Pbil

Population-based incremental learning.

· class Population

Population.

class ProgressTrackerContext

Log context for ProgressTracker.

• class PvAlgorithm

Probability vector algorithm.

• class RandomLocalSearch

Random local search.

• class RandomSearch

Random search.

class RandomWalk

Random walk.

class Restart
 Restart.

· class SimulatedAnnealing

Simulated annealing.

• class SteepestAscentHillClimbing

Steepest ascent hill climbing.

• class TournamentSelection

Population with tournament selection.

· class Umda

Univariate marginal distribution algorithm.

· class UniformCrossover

Uniform crossover.

Functions

```
    template < class T >

      bool matrix is symmetric (const std::vector< std::vector< T > > &A)
          Check for symmetric matrix.

    template<class T >

      bool matrix_is_strictly_lower_triangular (const std::vector< std::vector< T >> &A)
          Check for strictly lower triangular matrix.

    template<class T >

      bool matrix has diagonal (const std::vector< std::vector< T > > &A, T x)
          Check for diagonal elements.

    template<class T >

      bool matrix_has_range (const std::vector< std::vector< T >> &A, T inf, T sup)
          Check for element range.

    template < class T >

      bool matrix_has_dominant_diagonal (const std::vector< std::vector< T >> &A)
          Check for element range.
Type and functions related to probability vectors
    typedef std::vector< double > pv_t
```

```
Probability vector type.

    double pv_entropy (const pv_t &pv)

      Entropy of a probability vector.

    void pv_sample (const pv_t &pv, bit_vector_t &x)

      Sample a bit vector.
```

void pv_uniform (pv_t &pv)

Probability vector of the uniform distribution.

void pv init (pv t &pv)

Initialize.

void pv_add (pv_t &pv, const bit_vector_t &x)

Accumulate a bit vector.

void pv_add (pv_t &pv, const bit_vector_t &x, double weight)

Accumulate a bit vector.

void pv_average (pv_t &pv, int count)

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 BitHerding

Herding with bit features.

struct BitMoment

Moment for bit features.

• class Hea

Herding evolutionary algorithm.

class SpinHerding

Herding with spin variables.

• struct SpinMoment

Moment for spin variables.

4.4.1 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

Real black box functions defined on bit vectors.

Namespaces

real

Real multivariate functions.

Classes

class AbstractLabs

Abstract class for low autocorrelation binary sequences.

· class AbstractMaxSat

Abstract class for MaxSat-like functions.

· class AdditiveGaussianNoise

Additive Gaussian Noise.

· class Cache

Cache.

· class CallCounter

Call counter.

class DeceptiveJump

Deceptive jump.

class EqualProducts

Equal products.

· class Factorization

Factorization.

class FourPeaks

Four Peaks.

· class Function

Function.

class FunctionController

Function controller.

class FunctionDecorator

Function decorator.

class FunctionMapComposition

Composition of a function and a map.

• class FunctionModifier

Function modifier.

· class FunctionPlugin

Function plugin.

· class Hiff

Hierarchical if and only if.

· class Jump

Jump.

class Labs

Low autocorrelation binary sequences.

· class LabsMeritFactor

Low autocorrelation binary sequences merit factor.

class LeadingOnes

Leading ones.

· class LinearFunction

Linear function.

class LongPath

Long path.

class MaxNae3Sat

Max not-all-equal 3SAT.

class MaxSat

MAX-SAT.

• class NearestNeighborIsingModel1

Nearest neighbor Ising model in one dimension.

• class NearestNeighborIsingModel2

Nearest neighbor Ising model in two dimensions.

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 ParsedModifier

Parsed modifier.

· class Plateau

Plateau.

class PriorNoise

Prior noise.

class ProgressTracker

ProgressTracker.

· class Qubo

Quadratic unconstrained binary optimization.

• class Ridge

Ridge.

· class SinusSummationCancellation

Summation cancellation with sinus.

class SixPeaks

Six Peaks.

class StopOnMaximum

Stop on maximum.

class StopOnTarget

Stop on target.

class SummationCancellation

Summation cancellation.

class Trap

Trap.

class WalshExpansion

Walsh expansion.

class WalshExpansion1

Walsh expansion of degree 1.

class WalshExpansion2

Walsh expansion of degree 2.

Functions

- std::ostream & operator<< (std::ostream &stream, const ProgressTracker::Event &event)
 Insert formatted output.
- bool bv_is_locally_maximal (const bit_vector_t &bv, Function &fn, hnco::neighborhood::Neighborhood!terator &it)

Check whether a bit vector is locally maximal.

• bool bv_is_globally_maximal (const bit_vector_t &bv, Function &fn)

Check whether a bit vector is globally maximal.

4.6.1 Detailed Description

Real black box functions defined on bit vectors.

4.7 hnco::function::real Namespace Reference

Real multivariate functions.

Classes

· class DyadicRealRepresentation

Dyadic real representation.

· class ParsedRealMultivariateFunction

Parsed real multivariate function.

· class RealMultivariateFunction

Real multivariate function.

· class RealMultivariateFunctionAdapter

Real multivariate function adapter.

class RealRepresentation

Real representation.

4.7.1 Detailed Description

Real multivariate functions.

4.8 hnco::neighborhood Namespace Reference

Neighborhoods for local search.

Classes

· class BernoulliProcess

Bernoulli process.

class HammingBall

Hamming ball.

• class HammingSphere

Hamming sphere.

· class HammingSphereIterator

Hamming sphere neighborhood iterator.

class MultiBitFlip

Multi bit flip.

class Neighborhood

Neighborhood.

• class NeighborhoodIterator

Neighborhood iterator.

· class SingleBitFlip

One bit neighborhood.

• class SingleBitFlipIterator

Single bit flip neighborhood iterator.

4.8.1 Detailed Description

Neighborhoods for local search.

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

4.9 hnco::random Namespace Reference

Pseudo random numbers.

Classes

• struct Random Random numbers.

4.9.1 Detailed Description

Pseudo random numbers.

Chapter 5

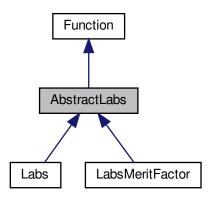
Class Documentation

5.1 AbstractLabs Class Reference

Abstract class for low autocorrelation binary sequences.

#include <hnco/functions/labs.hh>

Inheritance diagram for AbstractLabs:



Public Member Functions

AbstractLabs (int n)

Constructor.

• int get_bv_size ()

Get bit vector size.

double compute_autocorrelation (const bit_vector_t &)

Compute autocorrelation.

40 Class Documentation

Protected Attributes

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

5.1.1 Detailed Description

Abstract class for low autocorrelation binary sequences.

Definition at line 32 of file labs.hh.

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

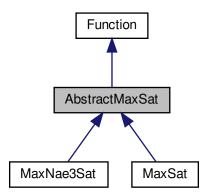
- · lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.2 AbstractMaxSat Class Reference

Abstract class for MaxSat-like functions.

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

Inheritance diagram for AbstractMaxSat:



Public Member Functions

AbstractMaxSat ()

Default constructor.

• int get_bv_size ()

Get bit vector size.

void display (std::ostream &stream)

Display the expression.

• virtual void load (std::istream &stream)

Load an instance.

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

Save an instance.

Protected Attributes

- std::vector< std::vector< int > > _expression
 Expression.
- int _num_variables

Number of variables.

5.2.1 Detailed Description

Abstract class for MaxSat-like functions.

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

5.2.2 Member Function Documentation

5.2.2.1 load()

Load an instance.

Exceptions

Error

Reimplemented in MaxNae3Sat.

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

5.2.3 Member Data Documentation

5.2.3.1 _expression

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

Expression.

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

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

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

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

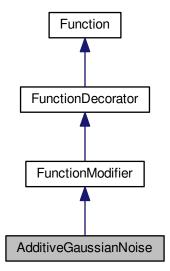
42 Class Documentation

5.3 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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

Inheritance diagram for AdditiveGaussianNoise:



Public Member Functions

- AdditiveGaussianNoise (Function *function, double stddev)
 Constructor.
- double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

• int get_bv_size ()

Get bit vector size.

Private Attributes

std::normal_distribution< double > _dist
 Normal distribution.

Additional Inherited Members

5.3.1 Detailed Description

Additive Gaussian Noise.

Definition at line 169 of file function-modifier.hh.

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

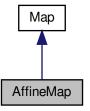
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

5.4 AffineMap Class Reference

Affine map.

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

Inheritance diagram for AffineMap:



Public Member Functions

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

Random instance.

void map (const bit_vector_t &input, bit_vector_t &output)

Mar

• int get_input_size ()

Get input size.

int get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

44 Class Documentation

Private Member Functions

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

Load.
```

Private Attributes

```
    bit_matrix_t _bm
        Bit matrix.
    bit_vector_t _bv
        Translation vector
```

Friends

· class boost::serialization::access

5.4.1 Detailed Description

Affine map.

void random (

Random instance.

An affine map f from F_2^m to F_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 258 of file map.hh.

5.4.2 Member Function Documentation

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

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

int rows, int cols,

bool surjective)

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:

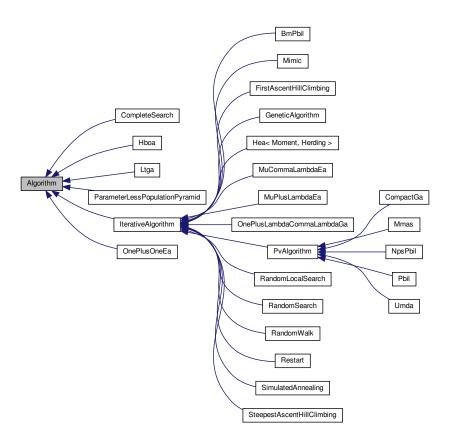
- · lib/hnco/map.hh
- lib/hnco/map.cc

5.5 Algorithm Class Reference

Abstract search algorithm.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for Algorithm:



46 Class Documentation

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 int get_bv_size ()

Get bit vector size.

Setters

```
    virtual void set_function (function::Function *function)
```

Set function.

virtual void set_functions (const std::vector< function::Function *> functions)

Set functions.

void set_stream (std::ostream *x)

Output stream.

void set_log_context (LogContext *lc)

Set log context.

Protected Member Functions

Managing solution

```
• void random_solution ()
```

Random solution.

void set_solution (const bit_vector_t &x, double value)

Set solution.

void set_solution (const bit_vector_t &x)

Set solution.

void update_solution (const bit_vector_t &x, double value)

Update solution (strict)

void update_solution (const point_value_t &pv)

Update solution (strict)

void update_solution (const bit_vector_t &x)

Update solution (strict).

Protected Attributes

```
    function::Function * _function
    std::vector< function::Function * > _functions
    Functions.
```

• point_value_t _solution

Solution

LogContext * _log_context = nullptr
 Log context.

Parameters

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

5.5.1 Detailed Description

Abstract search algorithm.

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

Definition at line 41 of file algorithm.hh.

5.5.2 Member Function Documentation

5.5.2.1 set_solution()

Set solution.

Warning

Evaluates the function once.

Definition at line 47 of file algorithm.cc.

48 Class Documentation

5.5.2.2 update_solution()

Update solution (strict).

Warning

Evaluates the function once.

Definition at line 70 of file algorithm.cc.

5.5.3 Member Data Documentation

5.5.3.1 _functions

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

Functions.

Each thread has its own function.

Definition at line 52 of file algorithm.hh.

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

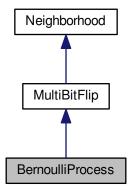
- · lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

5.6 BernoulliProcess Class Reference

Bernoulli process.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for BernoulliProcess:



Public Member Functions

```
• BernoulliProcess (int n)
```

Constructor.

• BernoulliProcess (int n, double p)

Constructor.

• void set_probability (double p)

Set probability.

Private Member Functions

```
• void sample_bits ()
```

Sample bits.

void bernoulli_process ()

Bernoulli process.

Private Attributes

```
• std::bernoulli_distribution _bernoulli_dist
```

Bernoulli distribution (biased coin)

• std::binomial_distribution< int > _binomial_dist

Binomial distribution.

bool <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

5.6.1 Detailed Description

Bernoulli process.

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

Definition at line 220 of file neighborhood.hh.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 BernoulliProcess() [1/2]

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

Constructor.

50 Class Documentation

Parameters

```
n Size of bit vectors
```

The Bernoulli probability is set to 1 / n.

Definition at line 255 of file neighborhood.hh.

5.6.2.2 BernoulliProcess() [2/2]

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

Constructor.

Parameters

n	Size of bit vectors
p	Bernoulli probability

Definition at line 265 of file neighborhood.hh.

5.6.3 Member Function Documentation

5.6.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

Definition at line 292 of file neighborhood.hh.

5.6.3.2 set_probability()

```
void set_probability ( \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:

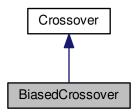
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.7 BiasedCrossover Class Reference

Biased crossover.

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

Inheritance diagram for BiasedCrossover:



Public Member Functions

• BiasedCrossover ()

Constructor.

- void breed (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)
 Breed.
- void set_bias (double b)
 Set bias.

Private Attributes

 std::bernoulli_distribution _bernoulli_dist Bernoulli distribution.

5.7.1 Detailed Description

Biased crossover.

Definition at line 75 of file crossover.hh.

5.7.2 Member Function Documentation

5.7.2.1 breed()

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

5.8 BitHerding Class Reference

Herding with bit features.

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

Public Types

• enum { DYNAMICS_MINIMIZE_NORM, DYNAMICS_MAXIMIZE_INNER_PRODUCT }

Public Member Functions

• BitHerding (int n)

Constructor.

• void init ()

Initialization.

void sample (const BitMoment &target, bit_vector_t &x)

Sample a bit vector.

double error (const BitMoment &target)

Compute the error.

Getters

const BitMoment & get_delta ()
 Get delta.

Setters

• void set_randomize_bit_order (bool x)

Randomize bit order.

void set_dynamics (int x)

Set the dynamics.

void set_weight (double x)

Set the weight of second order moments.

Protected Member Functions

void compute_delta (const BitMoment &target)

Compute delta.

void sample_minimize_norm (const BitMoment &target, bit_vector_t &x)

Sample a bit vector.

void sample_maximize_inner_product (const BitMoment &target, bit_vector_t &x)

Sample a bit vector.

Protected Attributes

· BitMoment _count

Counter moment.

· BitMoment _delta

Delta moment.

• permutation_t _permutation

Permutation.

• std::uniform_int_distribution< int > _choose_bit

Choose bit.

int _time

Time.

Parameters

• bool randomize bit order = false

Randomize bit order.

• int _dynamics = DYNAMICS_MINIMIZE_NORM

Dynamics.

• double _weight = 1

Weight of second order moments.

5.8.1 Detailed Description

Herding with bit features.

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

5.8.2 Member Enumeration Documentation

5.8.2.1 anonymous enum

anonymous enum

Enumerator

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 bit-herding.hh.

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

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

5.9 BitMoment Struct Reference

Moment for bit features.

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

Public Member Functions

• BitMoment (int n)

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

· void init ()

Initialize.

void add (const bit_vector_t &x)

Accumulate a bit vector.

· void average (int count)

Compute average.

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

Update moment.

void bound (double margin)

Bound moment.

· double distance (const BitMoment &p) const

Distance.

• double norm_2 () const

Compute the norm 2.

• double diameter () const

Compute the diameter.

• size_t size () const

Size.

· void display (std::ostream &stream)

Display.

Public Attributes

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

Moment.

• double _weight = 1

Weight of second order moments.

5.9.1 Detailed Description

Moment for bit features.

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

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

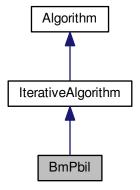
- · lib/hnco/algorithms/hea/bit-moment.hh
- lib/hnco/algorithms/hea/bit-moment.cc

5.10 BmPbil Class Reference

Boltzmann machine PBIL.

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

Inheritance diagram for BmPbil:



Public Types

- enum { LOG_NORM_INFINITE, LOG_NORM_L1, LAST_LOG }
- enum { RESET_NO_RESET, RESET_ITERATION, RESET_BIT_VECTOR }
- typedef std::bitset< LAST_LOG > log_flags_t

Public Member Functions

- BmPbil (int n, int population_size)
 - Constructor.
- · void init ()

Initialization.

Private Member Functions

· void iterate ()

Single iteration.

• void log ()

Log.

void sample (bit_vector_t &x)

Sample a bit vector.

• void sample_asynchronous ()

Asynchronous sampling.

• void sample_asynchronous_full_scan ()

Asynchronous sampling with full scan.

• void sample_synchronous ()

Synchronous sampling.

Private Attributes

• log_flags_t_log_flags

Log flags.

Population _population

Population.

· Model _model

Model.

ModelParameters _parameters_all

Parameters averaged over all individuals.

ModelParameters _parameters_best

Parameters averaged over selected individuals.

ModelParameters _parameters_worst
 Parameters averaged over negatively selected individuals.

• std::uniform_int_distribution< size_t > _choose_bit

Uniform distribution on bit_vector_t components.

• permutation_t _permutation

Permutation.

Parameters

• int _selection_size = 1

Selection size (number of selected individuals in the population)

• double _learning_rate = 1e-3

Learning rate.

int _num_gs_steps = 100

Number of gibbs sampler steps.

• int _num_gs_cycles = 1

Number of gibbs sampler cycles.

• bool _negative_positive_selection = false

Negative and positive selection.

• int sampling = SAMPLING ASYNCHRONOUS

Sampling mode.

int _mc_reset_strategy = RESET_NO_RESET

MC reset strategy.

void set_selection_size (int x)

Set the selection size.

• void set_learning_rate (double x)

Set the learning rate.

void set_num_gs_steps (int x)

Set the number of gibbs sampler steps.

• void set_num_gs_cycles (int x)

Set the number of gibbs sampler cycles.

• void set_negative_positive_selection (bool x)

Set negative and positive selection.

void set_sampling (int x)

Set the sampling mode.

void set_mc_reset_strategy (int x)

Set the MC reset strategy.

void set_log_flags (const log_flags_t &lf)

Set log flags.

Additional Inherited Members

5.10.1 Detailed Description

Boltzmann machine PBIL.

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

Reference:

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

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

5.10.2 Member Enumeration Documentation

5.10.2.1 anonymous enum

anonymous enum

Enumerator

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

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

5.10.2.2 anonymous enum

anonymous enum

Enumerator

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

5.10.2.3 anonymous enum

anonymous enum

Enumerator

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

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

5.10.3 Member Function Documentation

5.10.3.1 set_selection_size()

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

Set the selection size.

59

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

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

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

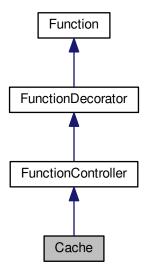
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

5.11 Cache Class Reference

Cache.

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

Inheritance diagram for Cache:



Public Member Functions

• Cache (Function *function)

Constructor.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

• double get_lookup_ratio ()

Get lookup ratio.

Evaluation

double eval (const bit_vector_t &)
 Evaluate a bit vector.

Private Attributes

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

Additional Inherited Members

5.11.1 Detailed Description

Cache.

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

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

Definition at line 362 of file function-controller.hh.

5.11.2 Constructor & Destructor Documentation

```
5.11.2.1 Cache()

Cache (

Function * function ) [inline]

Constructor.

Parameters

function | Decorated function |
```

Definition at line 381 of file function-controller.hh.

5.11.3 Member Function Documentation

5.11.3.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from FunctionController.

Definition at line 390 of file function-controller.hh.

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

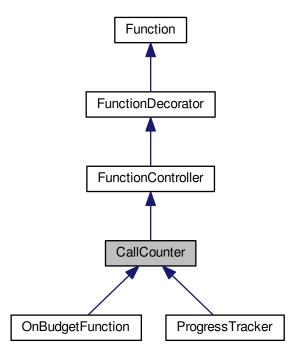
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.12 CallCounter Class Reference

Call counter.

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

Inheritance diagram for CallCounter:



Public Member Functions

• CallCounter (Function *function)

Constructor.

• int get_num_calls ()

Get the number of calls.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

double incremental_eval (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped
 — bits)

Incremental evaluation.

void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Protected Attributes

int _num_calls

Number of calls.

5.12.1 Detailed Description

Call counter.

Definition at line 173 of file function-controller.hh.

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

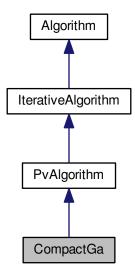
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.13 CompactGa Class Reference

Compact genetic algorithm.

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

Inheritance diagram for CompactGa:



Public Member Functions

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

Initialization.

Setters

• void set_learning_rate (double x) Set the learning rate.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

Parameters

• double <u>learning_rate</u> = 1e-3 *Learning rate*.

5.13.1 Detailed Description

Compact genetic algorithm.

Reference:

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

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

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

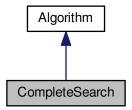
- lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

5.14 CompleteSearch Class Reference

Complete search.

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

Inheritance diagram for CompleteSearch:



Public Member Functions

• CompleteSearch (int n)

Constructor.

• void maximize ()

Maximize.

Additional Inherited Members

5.14.1 Detailed Description

Complete search.

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

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

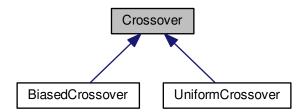
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

5.15 Crossover Class Reference

Crossover.

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

Inheritance diagram for Crossover:



Public Member Functions

- virtual ∼Crossover ()
 - Destructor.
- virtual void breed (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)=0
 Breed.

5.15.1 Detailed Description

Crossover.

Definition at line 35 of file crossover.hh.

5.15.2 Member Function Documentation

5.15.2.1 breed()

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:

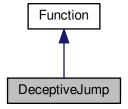
· lib/hnco/algorithms/ea/crossover.hh

5.16 DeceptiveJump Class Reference

Deceptive jump.

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

Inheritance diagram for DeceptiveJump:



Public Member Functions

• DeceptiveJump (int bv_size, int gap)

Constructor.

• int 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

```
• int _bv_size
```

Bit vector size.

int _gap

Gap.

5.16.1 Detailed Description

Deceptive jump.

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

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 85 of file jump.hh.

5.16.2 Member Function Documentation

```
5.16.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
_bv_size + _gap
```

Reimplemented from Function.

Definition at line 111 of file jump.hh.

5.16.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 107 of file jump.hh.

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

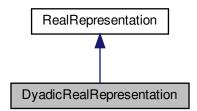
- · lib/hnco/functions/jump.hh
- · lib/hnco/functions/jump.cc

5.17 DyadicRealRepresentation Class Reference

Dyadic real representation.

#include <hnco/functions/real/real-representation.hh>

Inheritance diagram for DyadicRealRepresentation:



Public Member Functions

- DyadicRealRepresentation (double lower_bound, double upper_bound, int num_bits)
 Constructor.
- int size ()

Size of the representation.

• double convert (hnco::bit_vector_t::const_iterator first, hnco::bit_vector_t::const_iterator last)

Convert a bit vector range into a double.

Private Member Functions

double affine_transformation (double x)
 Affine transformation.

Private Attributes

```
    std::vector< double > _lengths
    Lengths of dyadic intervals.
```

• double _lower_bound

Lower bound of the search interval.

• double _length

Length of the search interval.

5.17.1 Detailed Description

Dyadic real representation.

Definition at line 52 of file real-representation.hh.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 DyadicRealRepresentation()

Constructor.

Parameters

lower_bound	Lower bound of the search interval
upper_bound	Upper bound of the search interval
num_bits	Number of bits per real

Definition at line 31 of file real-representation.cc.

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

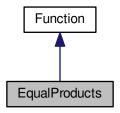
- · lib/hnco/functions/real/real-representation.hh
- lib/hnco/functions/real/real-representation.cc

5.18 EqualProducts Class Reference

Equal products.

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

Inheritance diagram for EqualProducts:



Public Member Functions

• EqualProducts ()

Constructor.

• int get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Instance generators

template < class Generator > void generate (int n, Generator generator)

Instance generator.

• void random (int n)

Random instance.

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

5.18.1 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.

5.18.2 Member Function Documentation

5.18.2.1 generate()

Instance generator.

Parameters

n	Size of bit vectors
generator	Number generator

Definition at line 94 of file equal-products.hh.

5.18.2.2 random()

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

Random instance.

The weights are sampled from the uniform distribution on [0,1).

5.19 Error Class Reference 73

Parameters

n Size of bit vector

Definition at line 109 of file equal-products.hh.

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

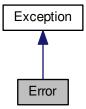
- · lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

5.19 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.

5.19.1 Detailed Description

Error.

Definition at line 84 of file exception.hh.

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

· lib/hnco/exception.hh

5.20 ProgressTracker::Event Struct Reference

Event.

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

Public Attributes

· int num_evaluations

Number of evaluations.

· double value

Value.

5.20.1 Detailed Description

Event.

Definition at line 222 of file function-controller.hh.

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

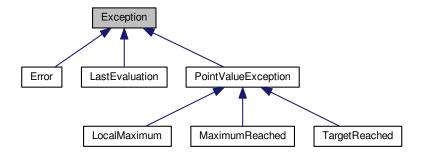
lib/hnco/functions/decorators/function-controller.hh

5.21 Exception Class Reference

Basic exception.

#include <hnco/exception.hh>

Inheritance diagram for Exception:



5.21.1 Detailed Description

Basic exception.

Definition at line 36 of file exception.hh.

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

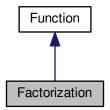
· lib/hnco/exception.hh

5.22 Factorization Class Reference

Factorization.

#include <hnco/functions/factorization.hh>

Inheritance diagram for Factorization:



Public Member Functions

· Factorization ()

Constructor.

• Factorization (const std::string number)

Constructor.

∼Factorization ()

Destructor.

void load (std::istream &stream)

Load an instance.

int 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 init ()

Init GMP data structures.

· void clear ()

Clear GMP data structures.

void set_number (const std::string number)

Set number.

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.

int _bv_size

Bit vector size.

5.22.1 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.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 Factorization()

Constructor.

Parameters

number	Number to factorize written in decimal form	
Hullibel	Number to factorize written in decimal form	

Definition at line 82 of file factorization.hh.

5.22.3 Member Function Documentation

5.22.3.1 load()

Load an instance.

Warning

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

Exceptions

Error

Definition at line 37 of file factorization.cc.

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

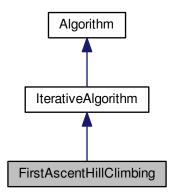
- lib/hnco/functions/factorization.hh
- lib/hnco/functions/factorization.cc

5.23 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

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

Inheritance diagram for FirstAscentHillClimbing:



Public Member Functions

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

Random initialization.

- void init (const bit_vector_t &x)
 - Explicit initialization.
- void init (const bit_vector_t &x, double value)

Explicit initialization.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

 neighborhood::NeighborhoodIterator * _neighborhood Neighborhood.

5.23.1 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:

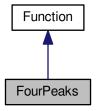
- · lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh
- · lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc

5.24 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.

• int 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

• int _bv_size

Bit vector size.

· int _threshold

Threshold.

• int _maximum

Maximum.

5.24.1 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.

5.24.2 Member Function Documentation

```
5.24.2.1 get_maximum()
```

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

Get the global maximum.

Returns

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

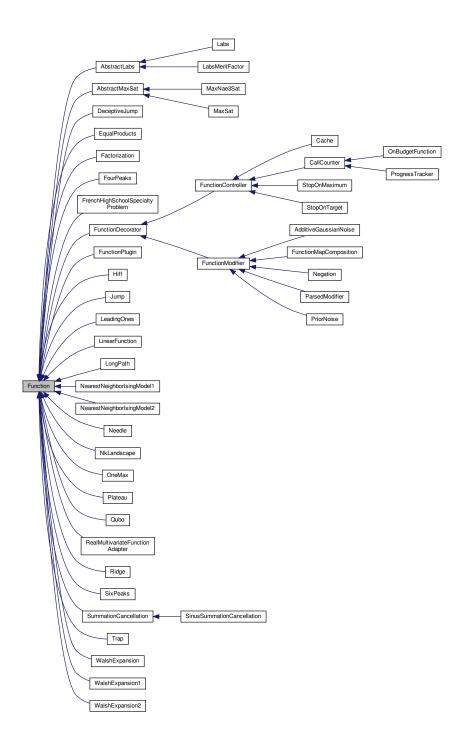
Reimplemented from Function.

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

5.24.2.2 has_known_maximum()
<pre>bool has_known_maximum () [inline], [virtual]</pre>
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.hhlib/hnco/functions/four-peaks.cc
5.25 Function Class Reference
Function.

#include <hnco/functions/function.hh>

Inheritance diagram for Function:



Classes

• struct WalshTransformTerm

Walsh transform term.

Public Member Functions

• virtual \sim Function ()

Destructor.

Information about the function

virtual int 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.

5.25.1 Detailed Description

Function.

Definition at line 41 of file function.hh.

5.25.2 Member Function Documentation

5.25.2.1 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 31 of file function.cc.

```
5.25.2.2 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented in Plateau, Ridge, Hiff, SixPeaks, Needle, FunctionMapComposition, WalshExpansion1, Leading Ones, LinearFunction, DeceptiveJump, LongPath, FourPeaks, SummationCancellation, Trap, PriorNoise, Jump, OneMax, and FunctionController.

Definition at line 80 of file function.hh.

5.25.2.3 incremental_eval()

Incremental evaluation.

Exceptions

Error

Reimplemented in OnBudgetFunction, ProgressTracker, CallCounter, NearestNeighborIsingModel2, Nearest↔ NeighborIsingModel1, StopOnTarget, StopOnMaximum, WalshExpansion1, LinearFunction, Negation, and One↔ Max.

Definition at line 133 of file function.hh.

5.25.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented in Cache, NearestNeighborlsingModel2, NearestNeighborlsingModel1, WalshExpansion1, LinearFunction, Negation, PriorNoise, OneMax, and FunctionController.

Definition at line 88 of file function.hh.

5.25.2.5 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 143 of file function.hh.

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

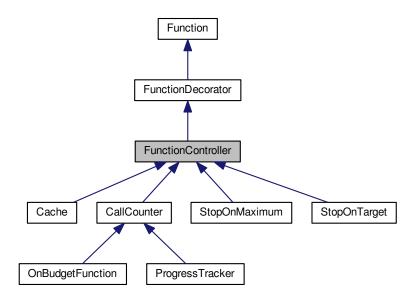
- lib/hnco/functions/function.hh
- · lib/hnco/functions/function.cc

5.26 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

• int get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

double safe_eval (const bit_vector_t &x)
 Safely evaluate a bit vector.

Additional Inherited Members

5.26.1 Detailed Description

Function controller.

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

5.26.2 Member Function Documentation

5.26.2.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true if the decorated function does

Reimplemented from Function.

Reimplemented in Cache.

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

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

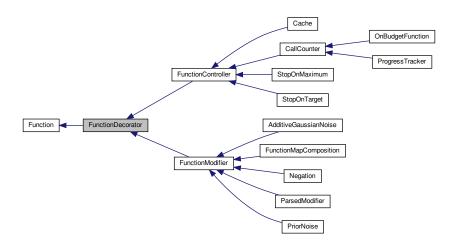
· lib/hnco/functions/decorators/function-controller.hh

5.27 FunctionDecorator Class Reference

Function decorator.

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

Inheritance diagram for FunctionDecorator:



Public Member Functions

• FunctionDecorator (Function *function)

Constructor.

Display

• void display (std::ostream &stream)

Display.

void describe (const bit_vector_t &x, std::ostream &stream)

Describe a bit vector.

Protected Attributes

• Function * _function

Decorated function.

5.27.1 Detailed Description

Function decorator.

Definition at line 33 of file function-decorator.hh.

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

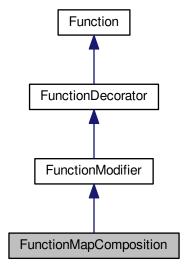
· lib/hnco/functions/decorators/function-decorator.hh

5.28 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

```
• int get_bv_size ()
```

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

Display

void describe (const bit_vector_t &x, std::ostream &stream)
 Describe a bit vector.

Private Attributes

```
    Map * _map
```

Мар.

bit_vector_t _bv

Image of bit vectors under the map.

Additional Inherited Members

5.28.1 Detailed Description

Composition of a function and a map.

Definition at line 99 of file function-modifier.hh.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 FunctionMapComposition()

Constructor.

Precondition

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

_			
Exce	ntı	α n	10
	บแ	vi	ı

Error

Definition at line 114 of file function-modifier.hh.

5.28.3 Member Function Documentation

```
5.28.3.1 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

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

5.28.3.2 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 144 of file function-modifier.hh.

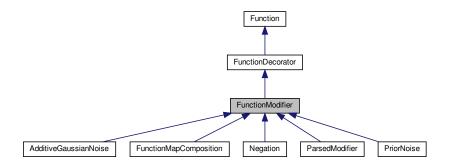
- · lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

5.29 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

5.29.1 Detailed Description

Function modifier.

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

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

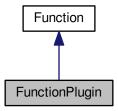
• lib/hnco/functions/decorators/function-modifier.hh

5.30 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.

• int 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 bit_t *, size_t)
 Type of an extern function.

Private Attributes

• int _bv_size

Bit vector size.

void * _handle

Handle returned by dlopen.

extern_function_t _extern_function

Extern function.

5.30.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 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 33 of file plugin.cc.

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

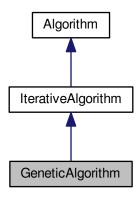
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

5.31 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



Public Member Functions

• GeneticAlgorithm (int n, int mu)

Constructor.

• void init ()

Initialization.

Setters

• void set_mutation_probability (double x)

Set the mutation probability.

• void set_crossover_probability (double x)

Set the crossover probability.

• void set_tournament_size (int x)

Set the tournament size.

void set_allow_stay (bool x)

Set the flag_allow_stay.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

• TournamentSelection _parents

Parents.

• TournamentSelection _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

• std::bernoulli_distribution _do_crossover

Do crossover.

• UniformCrossover _crossover

Uniform crossover.

Parameters

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

• double _crossover_probability = 0.5

Crossover probability.

• int _tournament_size = 10

Tournament size.

 bool _allow_stay = false Allow stay.

Additional Inherited Members

5.31.1 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.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 GeneticAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors
mu	Population size

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

5.31.3 Member Function Documentation

5.31.3.1 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:

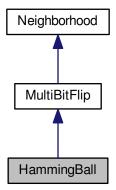
- · lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

5.32 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

std::uniform_int_distribution < int > _choose_k
 Choose the distance to the center.

Additional Inherited Members

5.32.1 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.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 HammingBall()

```
HammingBall (  \qquad \qquad \text{int } n, \\ \qquad \qquad \text{int } r \text{ ) } \quad [\text{inline}]
```

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the ball

Definition at line 320 of file neighborhood.hh.

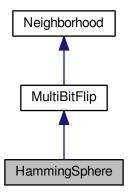
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.33 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

5.33.1 Detailed Description

Hamming sphere.

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

Definition at line 337 of file neighborhood.hh.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 HammingSphere()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the sphere

Definition at line 353 of file neighborhood.hh.

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

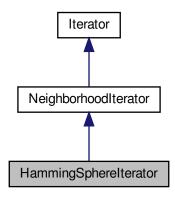
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.34 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

5.34.1 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.

5.35 Hboa Class Reference 101

5.34.2 Constructor & Destructor Documentation

5.34.2.1 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:

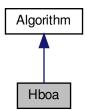
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.35 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

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

Inheritance diagram for Hboa:



Public Member Functions

• Hboa (int n)

Constructor.

• void maximize ()

Maximize.

void set_population_size (int n)

Set population size.

Private Attributes

int _population_size = 10
 Population size.

Additional Inherited Members

5.35.1 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 44 of file hboa.hh.

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

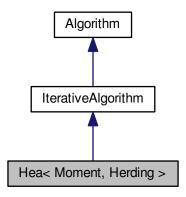
- · lib/hnco/algorithms/eda/hboa.hh
- · lib/hnco/algorithms/eda/hboa.cc

5.36 Hea< Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



Public Types

- enum {
 LOG_ERROR, LOG_DTU, LOG_DELTA, LOG_SELECTION,
 LOG_MOMENT_MATRIX, LAST_LOG }
- typedef std::bitset < LAST_LOG > log_flags_t
 Type for log flags.

Public Member Functions

- Hea (int n, int population_size)
 - Constructor.
- void init ()

Initialization.

Setters

- void set_herding (Herding *x)
 - Set the herding algorithm.
- void set_margin (double x)

Set the moment margin.

• void set_selection_size (int x)

Set the selection size.

void set_reset_period (int x)

Set the reset period.

void set_learning_rate (double x)

Set the learning rate.

void set_bound_moment (bool x)

Set the bound moment after update.

void set_weight (double weight)

Set weight.

void set_log_flags (const log_flags_t &lf)
 Set log flags.

Private Member Functions

```
• void iterate ()
```

Single iteration.

• void log ()

Log.

Private Attributes

Moment _target

Moment.

• Moment _selection

Moment of selected individuals.

• Moment <u>uniform</u>

Uniform moment.

• algorithm::Population _population

Population.

Herding * _herding

Herding.

Logging

• double _error_cache

Error cache.

· double dtu cache

Distance to uniform cache.

• double _delta_cache

Delta cache.

• double _selection_cache

Selection distance cache.

• log_flags_t _log_flags

Log flags.

Parameters

• double _margin

Moment margin.

• int _selection_size = 1

Selection size.

• int _reset_period = 0

Reset period.

• double <u>learning_rate</u> = 1e-4

Learning rate.

• bool _bound_moment = false

Bound moment after update.

Additional Inherited Members

5.36.1 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 50 of file hea.hh.

5.36.2 Member Enumeration Documentation

5.36.2.1 anonymous enum

anonymous enum

Enumerator

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

Definition at line 55 of file hea.hh.

5.36.3 Constructor & Destructor Documentation

Constructor.

Parameters

n	Size of bit vectors
population_size	Population size

_margin is initialized to 1 / n.

Definition at line 214 of file hea.hh.

5.36.4 Member Function Documentation

5.36.4.1 set_reset_period()

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

Set the reset period.

Parameters

Χ	Reset period
---	--------------

 $x \le 0$ means no reset.

Definition at line 258 of file hea.hh.

5.36.4.2 set_selection_size()

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

Set the selection size.

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

Definition at line 250 of file hea.hh.

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

• lib/hnco/algorithms/hea/hea.hh

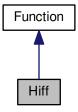
5.37 Hiff Class Reference 107

5.37 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.

• int 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

• int _bv_size

Bit vector size.

int _depth

Tree depth.

5.37.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 170 of file theory.hh.

5.37.2 Member Function Documentation

5.37.2.1 get_maximum() $\begin{tabular}{ll} double $\gcd_maximum()$ & [inline], [virtual] \\ \begin{tabular}{ll} Get the global maximum. \\ \begin{tabular}{ll} Returns & \\ & (i+1)*2^i where $2^i = bv_size \\ \end{tabular}$

Reimplemented from Function.

Definition at line 196 of file theory.hh.

5.37.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 192 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

5.38 HncoEvaluator Class Reference

Evaluator for HNCO functions.

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

Inheritance diagram for HncoEvaluator:



Public Member Functions

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

Constructor.

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

Evaluate a bit vector.

Private Attributes

• hnco::function::Function * _function

HNCO function.

hnco::bit_vector_t _bv

Argument of HNCO function.

5.38.1 Detailed Description

Evaluator for HNCO functions.

Definition at line 36 of file hnco-evaluator.hh.

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

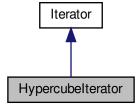
• lib/hnco/algorithms/eda/hnco-evaluator.hh

5.39 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

5.39.1 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:

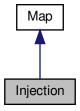
- · lib/hnco/iterator.hh
- lib/hnco/iterator.cc

5.40 Injection Class Reference

Injection.

#include <hnco/map.hh>

Inheritance diagram for Injection:



Public Member Functions

```
    Injection (const std::vector< int > &bit_positions, int output_size)
    Constructor.
```

void map (const bit_vector_t &input, bit_vector_t &output)

Мар

• int get_input_size ()

Get input size.

• int get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

• std::vector< int > _bit_positions

Bit positions.

· int _output_size

Output size.

5.40.1 Detailed Description

Injection.

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

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

An injection f from F_2^m to F_2^n , where $n \ge m$, is defined by f(x) = y, where, for all $j \in \{1, 2, \dots, m\}$, $y_{i_j} = x_j$.

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

Definition at line 397 of file map.hh.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Injection()

Constructor.

The input size of the map is given by the size of bit_positions.

Parameters

bit_positions	Bit positions in the output to where input bits are copied
output_size	Output size

Precondition

output_size >= bit_positions.size()

Definition at line 144 of file map.cc.

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

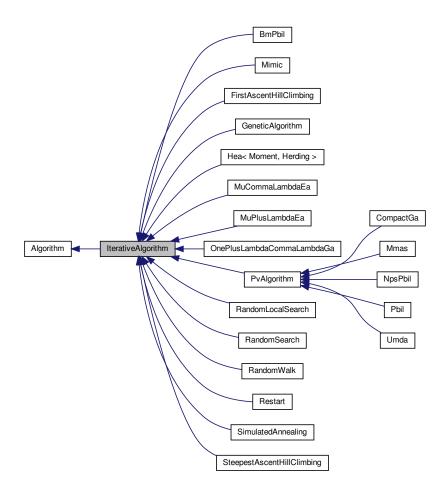
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.41 IterativeAlgorithm Class Reference

Iterative search.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for IterativeAlgorithm:



Public Member Functions

• IterativeAlgorithm (int n)

Constructor.

• void maximize ()

Maximize.

Setters

void set_num_iterations (int x)
 Set the number of iterations.

Protected Member Functions

```
    virtual void iterate ()=0
    Single iteration.
```

virtual void log ()
 Log.

Protected Attributes

· int _iteration

Current iteration.

• bool _something_to_log Something to log.

Parameters

• int _num_iterations = 0 Number of iterations.

5.41.1 Detailed Description

Iterative search.

Definition at line 169 of file algorithm.hh.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 IterativeAlgorithm()

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

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 199 of file algorithm.hh.

5.41.3 Member Function Documentation

5.41.3.1 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.

5.41.3.2 set_num_iterations()

Set the number of iterations.

Parameters

x Number of iterations

 $x \le 0$ means indefinite

Definition at line 223 of file algorithm.hh.

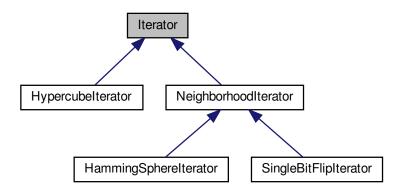
- · lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

5.42 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 <u>_initial_state</u> = true

Flag for initial state.

5.42.1 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:

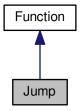
· lib/hnco/iterator.hh

5.43 Jump Class Reference

Jump.

#include <hnco/functions/jump.hh>

Inheritance diagram for Jump:



Public Member Functions

• Jump (int bv_size, int gap)

Constructor.

• int 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

• int _bv_size

Bit vector size.

int _gap

Gap.

5.43.1 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 41 of file jump.hh.

5.43.2 Member Function Documentation

```
5.43.2.1 get_maximum()

double get_maximum ( ) [inline], [virtual]

Get the global maximum.
```

Returns

_bv_size

Reimplemented from Function.

Definition at line 67 of file jump.hh.

```
5.43.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 63 of file jump.hh.

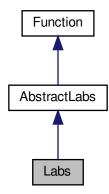
- lib/hnco/functions/jump.hh
- · lib/hnco/functions/jump.cc

5.44 Labs Class Reference

Low autocorrelation binary sequences.

#include <hnco/functions/labs.hh>

Inheritance diagram for Labs:



Public Member Functions

• Labs (int n)

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.44.1 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 65 of file labs.hh.

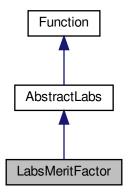
- · lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.45 LabsMeritFactor Class Reference

Low autocorrelation binary sequences merit factor.

#include <hnco/functions/labs.hh>

Inheritance diagram for LabsMeritFactor:



Public Member Functions

- LabsMeritFactor (int n)
 - Constructor.
- double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.45.1 Detailed Description

Low autocorrelation binary sequences merit factor.

Reference:

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

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

Definition at line 90 of file labs.hh.

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

5.46 LastEvaluation Class Reference

Last evaluation.

#include <hnco/exception.hh>

Inheritance diagram for LastEvaluation:



5.46.1 Detailed Description

Last evaluation.

Definition at line 80 of file exception.hh.

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

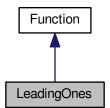
• lib/hnco/exception.hh

5.47 LeadingOnes Class Reference

Leading ones.

#include <hnco/functions/theory.hh>

Inheritance diagram for LeadingOnes:



Public Member Functions

• LeadingOnes (int bv_size)

Constructor.

• int 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

· int _bv_size

Bit vector size.

5.47.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 98 of file theory.hh.

5.47.2 Member Function Documentation

```
5.47.2.1 get_maximum()
```

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 122 of file theory.hh.

5.47.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 118 of file theory.hh.

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

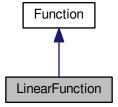
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.48 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



Public Member Functions

LinearFunction ()

Constructor.

Instance generators

template < class Generator >
 void generate (int n, Generator generator)
 Instance generator.

void random (int n)

Random instance.

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.

Information about the function

• int 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.

Private Member Functions

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

Private Attributes

std::vector< double > _weightsWeights.

Friends

· class boost::serialization::access

5.48.1 Detailed Description

Linear function.

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

5.48.2 Member Function Documentation

5.48.2.1 generate()

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

Definition at line 72 of file linear-function.hh.

```
5.48.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 119 of file linear-function.hh.

5.48.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 124 of file linear-function.hh.

5.48.2.4 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

n Size of bit vectors

Definition at line 86 of file linear-function.hh.

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

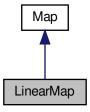
- · lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

5.49 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

• int get_input_size ()

Get input size.

int 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

5.49.1 Detailed Description

Linear map.

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

Definition at line 194 of file map.hh.

5.49.2 Member Function Documentation

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

Returns
    true if rank(_bm) == bm_num_rows(_bm)
Reimplemented from Map.
```

5.49.2.2 random()

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

Definition at line 90 of file map.cc.

Random instance.

Parameters

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

Exceptions



Definition at line 61 of file map.cc.

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

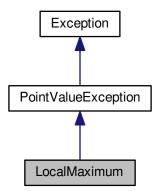
- · lib/hnco/map.hh
- lib/hnco/map.cc

5.50 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

5.50.1 Detailed Description

Local maximum.

Definition at line 71 of file exception.hh.

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

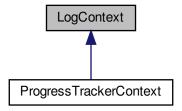
· lib/hnco/exception.hh

5.51 LogContext Class Reference

Log context.

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

Inheritance diagram for LogContext:



Public Member Functions

 virtual std::string get_context ()=0
 Get context.

5.51.1 Detailed Description

Log context.

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

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

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

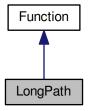
• lib/hnco/algorithms/log-context.hh

5.52 LongPath Class Reference

Long path.

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

Inheritance diagram for LongPath:



Public Member Functions

- LongPath (int bv_size, int prefix_length)
 - Constructor.
- double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

- int get_bv_size ()
 - Get bit vector size.
- bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

- int _bv_size
 - Bit vector size.
- · int _prefix_length

Prefix length.

5.52.1 Detailed Description

Long path.

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

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

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

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

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

References:

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

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.52.2 Member Function Documentation

5.52.2.1 get_maximum()

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

Get the global maximum.

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

Exceptions

Error

Reimplemented from Function.

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

5.52.2.2 has_known_maximum()

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

Check for a known maximum.

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

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

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

Reimplemented from Function.

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

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

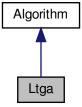
- · lib/hnco/functions/long-path.hh
- · lib/hnco/functions/long-path.cc

5.53 Ltga Class Reference

Linkage Tree Genetic Algorithm.

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

Inheritance diagram for Ltga:



Public Member Functions

• Ltga (int n)

Constructor.

• void maximize ()

Maximize.

void set_population_size (int n)

Set population size.

Private Attributes

int _population_size = 10Population size.

Additional Inherited Members

5.53.1 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 42 of file Itga.hh.

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

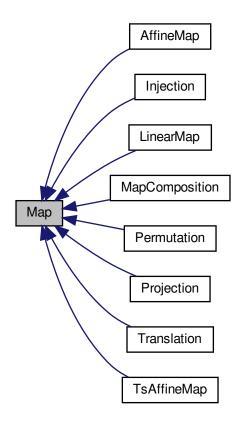
- lib/hnco/algorithms/eda/ltga.hh
- lib/hnco/algorithms/eda/ltga.cc

5.54 Map Class Reference

Мар

#include <hnco/map.hh>

Inheritance diagram for Map:



Public Member Functions

- virtual \sim Map ()
 - Destructor.
- virtual void map (const bit_vector_t &input, bit_vector_t &output)=0

Мар

• virtual int get_input_size ()=0

Get input size.

• virtual int get_output_size ()=0

Get output size.

• virtual bool is_surjective ()

Check for surjective map.

5.54.1 Detailed Description

Мар

Definition at line 40 of file map.hh.

5.54.2 Member Function Documentation

5.54.2.1 is_surjective()

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

Check for surjective map.

Returns

false

Reimplemented in TsAffineMap, Projection, Injection, MapComposition, AffineMap, LinearMap, Permutation, and Translation.

Definition at line 60 of file map.hh.

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

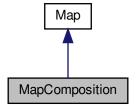
• lib/hnco/map.hh

5.55 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)

Мар

• int get_input_size ()

Get input size.

• int 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.

5.55.1 Detailed Description

Map composition.

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

Definition at line 328 of file map.hh.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 MapComposition()

Constructor.

Parameters

outer	outer map
inner	inner map

Precondition

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

Definition at line 352 of file map.hh.

5.55.3 Member Function Documentation

5.55.3.1 is_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 376 of file map.hh.

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

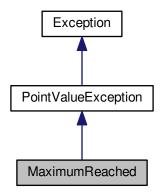
· lib/hnco/map.hh

5.56 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

5.56.1 Detailed Description

Maximum reached.

Definition at line 53 of file exception.hh.

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

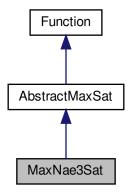
· lib/hnco/exception.hh

5.57 MaxNae3Sat Class Reference

Max not-all-equal 3SAT.

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

Inheritance diagram for MaxNae3Sat:



Public Member Functions

MaxNae3Sat ()

Default constructor.

void load (std::istream &stream)

Load an instance.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.57.1 Detailed Description

Max not-all-equal 3SAT.

Reference:

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

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

5.57.2 Member Function Documentation

```
5.57.2.1 load()
```

Load an instance.

Exceptions

Error

Reimplemented from AbstractMaxSat.

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

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

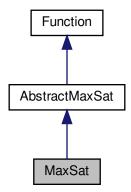
- · lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.58 MaxSat Class Reference

MAX-SAT.

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

Inheritance diagram for MaxSat:



Public Member Functions

• MaxSat ()

Default constructor.

• void random (int n, int k, int c)

Random instance.

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

Random instance with satisfiable expression.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.58.1 Detailed Description

MAX-SAT.

Reference:

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

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

5.58.2 Member Function Documentation

```
5.58.2.1 random() [1/2]
void random (
int n,
int k,
int c)
```

Random instance.

Parameters

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

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

Random instance with satisfiable expression.

Warning

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

Parameters

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

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

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

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

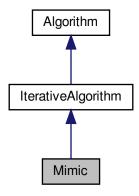
5.59 Mimic Class Reference

Mutual information maximizing input clustering.

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

5.59 Mimic Class Reference 141

Inheritance diagram for Mimic:



Public Member Functions

- Mimic (int n, int population_size)
 - Constructor.
- void init ()

Initialization.

Setters

void set_selection_size (int x)
 Set the selection size.

Protected Member Functions

• void iterate ()

Single iteration.

void sample (bit_vector_t &bv)

Sample a bit vector.

void compute_conditional_entropy (int index)

Compute conditional entropy.

• void update_model ()

Update model.

Protected Attributes

• Population _population

Population.

· permutation_t _permutation

Permutation.

std::array< pv_t, 2 > _parameters

Model parameters.

pv_t _mean

Mean of selected bit vectors.

std::vector< double > _entropies

Conditional entropies.

std::array< std::array< int, 2 >, 2 > _table

Contingency table.

• double _lower_bound

Lower bound of probability.

• double <u>upper_bound</u>

Upper bound of probability.

Parameters

• int _selection_size Selection size.

5.59.1 Detailed Description

Mutual information maximizing input clustering.

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

Reference:

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

Definition at line 54 of file mimic.hh.

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

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

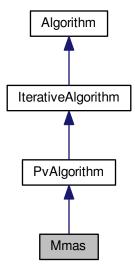
5.60 Mmas Class Reference 143

5.60 Mmas Class Reference

Max-min ant system.

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

Inheritance diagram for Mmas:



Public Member Functions

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

Initialization.

Setters

- void set_compare (std::function< bool(double, double)> x)
 Set the binary operator for comparing evaluations.
- void set_learning_rate (double x) Set the learning rate.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

bit_vector_t _x

Candidate solution.

Parameters

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

double <u>learning_rate</u> = 1e-3
 Learning rate.

5.60.1 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 42 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

5.61 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.

5.61.1 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

5.62 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

5.62.1 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:

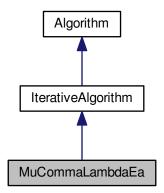
- · lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

5.63 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



Public Member Functions

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

Constructor.

· void init ()

Initialization.

Setters

- void set_mutation_probability (double x)
 - Set the mutation probability.
- void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

• void iterate ()

Single iteration.

Private Attributes

· Population _parents

Parents.

• Population _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

std::uniform_int_distribution< int > _select_parent

Select parent.

Parameters

• double _mutation_probability

Mutation probability.

bool _allow_stay = false

Allow stay.

Additional Inherited Members

5.63.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.63.2 Constructor & Destructor Documentation

5.63.2.1 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.

5.63.3 Member Function Documentation

5.63.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

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

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

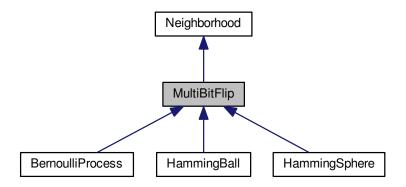
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

5.64 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

5.64.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

5.64.2 Constructor & Destructor Documentation

5.64.2.1 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.

5.64.3 Member Function Documentation

5.64.3.1 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.

5.64.3.2 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:

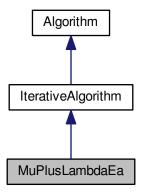
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.65 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

5.65.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.65.2 Constructor & Destructor Documentation

5.65.2.1 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.

5.65.3 Member Function Documentation

5.65.3.1 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:

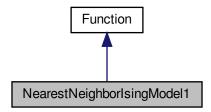
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

5.66 NearestNeighborlsingModel1 Class Reference

Nearest neighbor Ising model in one dimension.

#include <hnco/functions/ising/nearest-neighbor-ising-model-1.hh>

Inheritance diagram for NearestNeighborlsingModel1:



Public Member Functions

• NearestNeighborIsingModel1 ()

Constructor.

• void set_periodic_boundary_conditions (bool x)

Set periodic boundary conditions.

void display (std::ostream &stream)

Display.

Instance generators

template < class CouplingGen, class FieldGen >
 void generate (int n, CouplingGen coupling_gen, FieldGen field_gen)
 Instance generator.

• void random (int n)

Random instance.

Evaluation

• double eval (const bit_vector_t &)

Evaluate a bit vector.

double incremental_eval (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits)
 Incremental evaluation.

Information about the function

• int get_bv_size ()

Get bit vector size.

bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Private Member Functions

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

• template < class Archive >

void load (Archive &ar, const unsigned int version)

Load.

• void resize (int n)

Resize data structures.

Private Attributes

std::vector< double > _coupling

Coupling with nearest neighbor to the right.

std::vector< double > _field

External field.

bit_vector_t _flipped_bits

Flipped bits.

• bool _periodic_boundary_conditions = false

Periodic boundary conditions.

Friends

· class boost::serialization::access

5.66.1 Detailed Description

Nearest neighbor Ising model in one dimension.

Its expression is of the form

$$f(x) = \sum_{i} J_{i,i+1}(1 - 2x_i)(1 - 2x_{i+1}) + \sum_{i} h_i(1 - 2x_i)$$

or equivalently

$$f(x) = \sum_{i} J_{i,i+1}(-1)^{x_i + x_{i+1}} + \sum_{i} h_i(-1)^{x_i}$$

where $J_{i,i+1}$ is the interaction between adjacent sites i and i+1 and h_i is the external magnetic field interacting with site i.

In the case of periodic boundary conditions, the sum i+1 is mod n.

Since we are maximizing f or minimizing -f, the expression of f is compatible with what can be found in physics textbooks.

It should be noted that such an Ising model can be represented by a Walsh expansion of degree 2, that is Walsh ← Expansion2.

Reference: https://en.wikipedia.org/wiki/Ising_model

Definition at line 65 of file nearest-neighbor-ising-model-1.hh.

5.66.2 Member Function Documentation

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

Definition at line 44 of file nearest-neighbor-ising-model-1.cc.

5.66.2.2 generate()

```
void generate (
          int n,
          CouplingGen coupling_gen,
          FieldGen field_gen ) [inline]
```

Instance generator.

Parameters

n	Size of bit vectors
coupling_gen	Coupling generator
field_gen	External field generator

Definition at line 126 of file nearest-neighbor-ising-model-1.hh.

5.66.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 176 of file nearest-neighbor-ising-model-1.hh.

5.66.2.4 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

```
n Size of bit vector
```

Definition at line 142 of file nearest-neighbor-ising-model-1.hh.

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

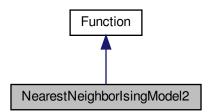
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.cc

5.67 NearestNeighborlsingModel2 Class Reference

Nearest neighbor Ising model in two dimensions.

#include <hnco/functions/ising/nearest-neighbor-ising-model-2.hh>

Inheritance diagram for NearestNeighborIsingModel2:



Public Member Functions

· NearestNeighborIsingModel2 ()

Constructor.

• void set_periodic_boundary_conditions (bool x)

Set periodic boundary conditions.

· void display (std::ostream &stream)

Display.

Instance generators

- template < class CouplingGen, class FieldGen >
 void generate (int num_rows, int num_columns, CouplingGen coupling_gen, FieldGen field_gen)
 Instance generator.
- void random (int num_rows, int num_columns)
 Random instance.

Evaluation

- double eval (const bit vector t &)
 - Evaluate a bit vector.
- double incremental_eval (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits)
 Incremental evaluation.

Information about the function

- int get_bv_size ()
 - Get bit vector size.
- bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

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.

· void resize (int num rows, int num columns)

Resize data structures.

Private Attributes

- std::vector< std::vector< double >> _coupling_right
 - Coupling with nearest neighbor to the right.
- std::vector< std::vector< double >> _coupling_below

Coupling with nearest neighbor below.

- std::vector< std::vector< double >> _field
 - External field.
- bit_vector_t _flipped_bits

Flipped bits.

• bool _periodic_boundary_conditions = false

Periodic boundary conditions.

Friends

· class boost::serialization::access

5.67.1 Detailed Description

Nearest neighbor Ising model in two dimensions.

We are considering a rectangular lattice in which each site has (at most) four neighbors (left, right, above, below).

The expression of the function is of the form

$$f(x) = \sum_{(i,j)} J_{ij}(1 - 2x_i)(1 - 2x_j) + \sum_i h_i(1 - 2x_i)$$

or equivalently

$$f(x) = \sum_{(i,j)} J_{ij}(-1)^{x_i + x_j} + \sum_i h_i(-1)^{x_i}$$

where the first sum is over adjacent sites (i, j), J_{ij} is the interaction between adjacent sites i and j, and h_i is the external magnetic field interacting with site i.

Since we are maximizing f or minimizing -f, the expression of f is compatible with what can be found in physics textbooks.

It should be noted that such an Ising model can be represented by a Walsh expansion of degree 2, that is Walsh ← Expansion2.

Reference: https://en.wikipedia.org/wiki/Ising_model

Definition at line 67 of file nearest-neighbor-ising-model-2.hh.

5.67.2 Member Function Documentation

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

Definition at line 47 of file nearest-neighbor-ising-model-2.cc.

5.67.2.2 generate()

```
void generate (
          int num_rows,
          int num_columns,
          CouplingGen coupling_gen,
          FieldGen field_gen ) [inline]
```

Instance generator.

Parameters

num_rows	Number of rows
num_columns	Number of columns
coupling_gen	Coupling generator
field_gen	External field generator

Definition at line 134 of file nearest-neighbor-ising-model-2.hh.

5.67.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 194 of file nearest-neighbor-ising-model-2.hh.

5.67.2.4 random()

```
void random (
                int num_rows,
                int num_columns ) [inline]
```

Random instance.

The weights are sampled from the normal distribution.

Parameters

num_rows	Number of rows
num_columns	Number of columns

Definition at line 154 of file nearest-neighbor-ising-model-2.hh.

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

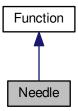
- · lib/hnco/functions/ising/nearest-neighbor-ising-model-2.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-2.cc

5.68 Needle Class Reference

Needle in a haystack.

#include <hnco/functions/theory.hh>

Inheritance diagram for Needle:



Public Member Functions

Needle (int bv_size)

 ${\it Constructor.}$

• int 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

• int _bv_size

Bit vector size.

5.68.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 134 of file theory.hh.

5.68.2 Member Function Documentation

```
5.68.2.1 get_maximum()
double get_maximum ( ) [inline], [virtual]
Get the global maximum.
Returns
     1
Reimplemented from Function.
Definition at line 158 of file theory.hh.
5.68.2.2 has_known_maximum()
bool has_known_maximum ( ) [inline], [virtual]
Check for a known maximum.
Returns
     true
Reimplemented from Function.
Definition at line 154 of file theory.hh.
```

· lib/hnco/functions/theory.hh

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

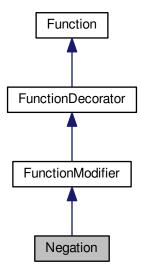
• lib/hnco/functions/theory.cc

5.69 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

• int get_bv_size ()

Get bit vector size.

• 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

5.69.1 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 59 of file function-modifier.hh.

5.69.2 Member Function Documentation

5.69.2.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 78 of file function-modifier.hh.

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

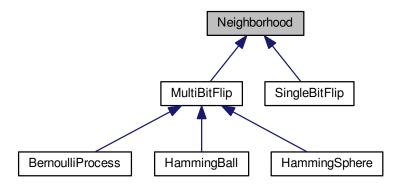
- · lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

5.70 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

 $\bullet \quad \mathsf{std}{::}\mathsf{uniform_int_distribution}{<} \ \mathsf{int} > \underline{\mathsf{index_dist}} \\$

Index distribution.

• sparse_bit_vector_t _flipped_bits

Flipped bits.

5.70.1 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.

5.70.2 Constructor & Destructor Documentation

5.70.2.1 Neighborhood()

```
Neighborhood ( \label{eq:neighborhood} \text{ int } n \text{ ) } \quad [\text{inline}]
```

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 86 of file neighborhood.hh.

5.70.3 Member Function Documentation

```
5.70.3.1 map()
```

Мар.

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

Parameters

input	Input bit vector
output	Output bit vector

Definition at line 148 of file neighborhood.hh.

5.70.3.2 mutate()

```
virtual void mutate ( bit\_vector\_t \ \& \ bv \ ) \quad [inline] \text{, [virtual]}
```

Mutate.

In-place mutation of the bit vector.

Parameters

```
bv Bit vector to mutate
```

Definition at line 134 of file neighborhood.hh.

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

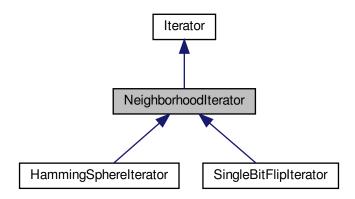
lib/hnco/neighborhoods/neighborhood.hh

5.71 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

5.71.1 Detailed Description

Neighborhood iterator.

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

5.71.2 Constructor & Destructor Documentation

5.71.2.1 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:

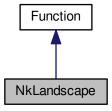
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.72 NkLandscape Class Reference

NK landscape.

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

Inheritance diagram for NkLandscape:



Public Member Functions

• NkLandscape ()

Default constructor.

• int get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

· void display (std::ostream &stream)

Display.

Instance generators

template < class Generator >
 void generate (int n, int k, Generator generator)
 Instance generator.

• void random (int n, int k)

Random instance.

Private Member Functions

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

• void random_structure (int n, int k)

Random structue.

Private Attributes

```
    std::vector < std::vector < int > > _neighbors
    Bit neighbors.
```

std::vector< std::vector< double >> _partial_functions
 Partial functions.

Friends

· class boost::serialization::access

5.72.1 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.

5.72.2 Member Function Documentation

5.72.2.1 generate()

Instance generator.

Parameters

n	Size of bit vector
k	Number of neighbors of each bit
generator	Generator for partial function values

Definition at line 92 of file nk-landscape.hh.

5.72.2.2 random()

Random instance.

Partial function values are sampled from the normal distribution.

Parameters

```
n Size of bit vector
```

Definition at line 109 of file nk-landscape.hh.

5.72.2.3 random_structure()

```
void random_structure (
          int n,
          int k) [private]
```

Random structue.

Parameters

r	,	Size of bit vector
k	,	Number of neighbors of each bit

Definition at line 32 of file nk-landscape.cc.

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

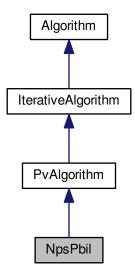
- lib/hnco/functions/nk-landscape.hh
- · lib/hnco/functions/nk-landscape.cc

5.73 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

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

Inheritance diagram for NpsPbil:



Public Member Functions

- NpsPbil (int n, int population_size)
 Constructor.
- void init ()
 Initialization.

Setters

- void set_selection_size (int x)
 - Set the selection size.
- void set_learning_rate (double x) Set the learning rate.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

- pv_t _mean_best

Mean of best individuals.

pv_t _mean_worst

Mean of worst individuals.

Parameters

int _selection_size = 1
 Selection size.
 double _learning_rate = 1e-3
 Learning rate.

5.73.1 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 42 of file nps-pbil.hh.

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

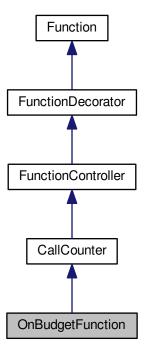
- lib/hnco/algorithms/pv/nps-pbil.hh
- lib/hnco/algorithms/pv/nps-pbil.cc

5.74 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

5.74.1 Detailed Description

CallCounter with a limited number of evaluations.

Definition at line 317 of file function-controller.hh.

5.74.2 Member Function Documentation

```
5.74.2.1 eval()
```

Evaluate a bit vector.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.74.2.2 incremental_eval()

Incremental evaluation.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.74.2.3 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:

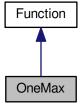
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.75 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

- int 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

int _bv_size

Bit vector size.

5.75.1 Detailed Description

OneMax.

References:

Heinz Mühlenbein, "How genetic algorithms really work: I. mutation and hillclimbing", in Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, 1992

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 41 of file theory.hh.

5.75.2 Member Function Documentation

```
5.75.2.1 get_maximum()
double get_maximum ( ) [inline], [virtual]
Get the global maximum.
Returns
     _bv_size
Reimplemented from Function.
Definition at line 62 of file theory.hh.
5.75.2.2 has_known_maximum()
bool has_known_maximum ( ) [inline], [virtual]
Check for a known maximum.
Returns
     true
Reimplemented from Function.
Definition at line 66 of file theory.hh.
5.75.2.3 provides_incremental_evaluation()
bool provides_incremental_evaluation ( ) [inline], [virtual]
Check whether the function provides incremental evaluation.
Returns
     true
Reimplemented from Function.
```

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

· lib/hnco/functions/theory.hh

Definition at line 71 of file theory.hh.

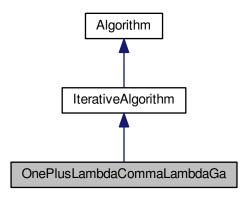
• lib/hnco/functions/theory.cc

5.76 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

5.76.1 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.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 OnePlusLambdaCommaLambdaGa()

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:

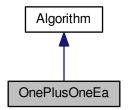
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

5.77 OnePlusOneEa Class Reference

(1+1) EA.

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

Inheritance diagram for OnePlusOneEa:



Public Member Functions

OnePlusOneEa (int n)

Constructor.

void set_function (function::Function *function)

Set function.

void init ()

Initialization.

• void maximize ()

Maximize.

const point_value_t & get_solution ()
 Solution.

Setters

void set_num_iterations (int x)

Set the number of iterations.

void set_mutation_probability (double x)

Set the mutation probability.

void set_allow_stay (bool x)

Set the flag _allow_stay.

void set_incremental_evaluation (bool x)

Set incremental evaluation.

Private Attributes

• neighborhood::BernoulliProcess _neighborhood

Neighborhood.

• RandomLocalSearch _rls

Random local search.

Parameters

```
• int _num_iterations = 0
```

Number of iterations.

· double _mutation_probability

Mutation probability.

• bool <u>_allow_stay</u> = false

Allow stay.

• bool _incremental_evaluation = false

Incremental evaluation.

Additional Inherited Members

5.77.1 Detailed Description

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

(1+1) EA is implemented as a RandomLocalSearch with a BernoulliProcess neighborhood and infinite patience. Thus it does derive from IterativeAlgorithm. It should be noted that member Algorithm::_solution is not used by OnePlusOneEa.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 45 of file one-plus-one-ea.hh.

5.77.2 Constructor & Destructor Documentation

5.77.2.1 OnePlusOneEa()

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

Constructor.

Parameters

n Size of bit vectors

_mutation_probability is initialized to 1 / n.

Definition at line 80 of file one-plus-one-ea.hh.

5.77.3 Member Function Documentation

5.77.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

Definition at line 127 of file one-plus-one-ea.hh.

5.77.3.2 set_num_iterations()

Set the number of iterations.

Parameters

```
x Number of iterations
```

 $x \le 0$ means indefinite

Definition at line 117 of file one-plus-one-ea.hh.

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

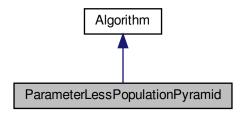
• lib/hnco/algorithms/ea/one-plus-one-ea.hh

5.78 ParameterLessPopulationPyramid Class Reference

Parameter-less Population Pyramid.

```
#include <hnco/algorithms/eda/p3.hh>
```

Inheritance diagram for ParameterLessPopulationPyramid:



Public Member Functions

ParameterLessPopulationPyramid (int n)

Constructor.

· void maximize ()

Maximize.

Additional Inherited Members

5.78.1 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 46 of file p3.hh.

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

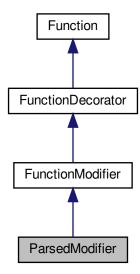
- · lib/hnco/algorithms/eda/p3.hh
- lib/hnco/algorithms/eda/p3.cc

5.79 ParsedModifier Class Reference

Parsed modifier.

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

Inheritance diagram for ParsedModifier:



Public Member Functions

ParsedModifier (Function *function, std::string expression)
 Constructor.

Information about the function

• int get_bv_size ()

Get bit vector size.

Evaluation

double eval (const bit_vector_t &)
 Evaluate a bit vector.

Private Attributes

• FunctionParser _fparser

Function parser.

• double _values [1]

Array of values.

Additional Inherited Members

5.79.1 Detailed Description

Parsed modifier.

Let f be the original function. Then the modified function is equivalent to $g \circ f$, where g is a real function defined by an expression g(x) provided as a string.

Definition at line 39 of file parsed-modifier.hh.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 ParsedModifier()

Constructor.

Parameters

function	Decorated function
expression	Expression to parse

Definition at line 30 of file parsed-modifier.cc.

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

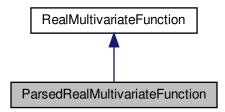
- · lib/hnco/functions/decorators/parsed-modifier.hh
- lib/hnco/functions/decorators/parsed-modifier.cc

5.80 ParsedRealMultivariateFunction Class Reference

Parsed real multivariate function.

#include <hnco/functions/real/real-multivariate-function.hh>

Inheritance diagram for ParsedRealMultivariateFunction:



Public Member Functions

• ParsedRealMultivariateFunction (std::string expression)

Constructor.

• int get dimension ()

Get the dimension of vectors.

double eval (const std::vector< double > x)

Evaluate a real vector.

Private Attributes

• FunctionParser _fparser

Function parser.

• int _num_variables = 0

Number of variables.

5.80.1 Detailed Description

Parsed real multivariate function.

Definition at line 52 of file real-multivariate-function.hh.

5.80.2 Constructor & Destructor Documentation

5.80.2.1 ParsedRealMultivariateFunction()

Constructor.

Parameters

expression Expression to

Definition at line 34 of file real-multivariate-function.cc.

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

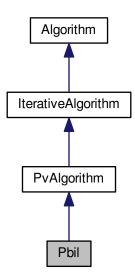
- · lib/hnco/functions/real/real-multivariate-function.hh
- lib/hnco/functions/real/real-multivariate-function.cc

5.81 Pbil Class Reference

Population-based incremental learning.

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

Inheritance diagram for Pbil:



Public Member Functions

- Pbil (int n, int population_size)
 - Constructor.
- void init ()

Initialization.

Setters

- void set_selection_size (int x)
 - Set the selection size.
- void set_learning_rate (double x)

Set the learning rate.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

pv_t _mean

Mean of selected bit vectors.

Parameters

int _selection_size = 1
 Selection size.
 double _learning_rate = 1e-3
 Learning rate.

5.81.1 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 41 of file pbil.hh.

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

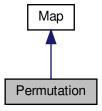
- · lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

5.82 Permutation Class Reference

Permutation.

#include <hnco/map.hh>

Inheritance diagram for Permutation:



Public Member Functions

```
• void random (int n)
```

Random instance.

void map (const bit_vector_t &input, bit_vector_t &output)

Map

• int get_input_size ()

Get input size.

• int 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

Friends

· class boost::serialization::access

5.82.1 Detailed Description

Permutation.

A permutation is a linear map f from F_2^n to itself defined by f(x)=y, where $y_i=x_{\sigma_i}$ and σ is a permutation of 0, 1, ..., n - 1.

Definition at line 133 of file map.hh.

5.82.2 Member Function Documentation

5.82.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 184 of file map.hh.

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

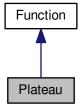
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.83 Plateau Class Reference

Plateau.

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

Inheritance diagram for Plateau:



Public Member Functions

• Plateau (int bv_size)

Constructor.

• int 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

```
• int _bv_size

Bit vector size.
```

5.83.1 Detailed Description

Plateau.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 244 of file theory.hh.

5.83.2 Member Function Documentation

```
5.83.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
_bv_size + 2
```

Reimplemented from Function.

Definition at line 268 of file theory.hh.

```
5.83.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 264 of file theory.hh.

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

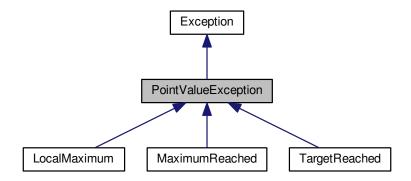
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.84 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 _pvPoint-value.

5.84.1 Detailed Description

Point-value exception.

Definition at line 39 of file exception.hh.

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

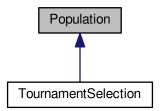
• lib/hnco/exception.hh

5.85 Population Class Reference

Population.

#include <hnco/algorithms/population.hh>

Inheritance diagram for Population:



Public Types

typedef std::pair< int, double > index_value_t
 Index-value type.

Public Member Functions

• Population (int population_size, int n)

Constructor.

· int size () const

Size.

• void random ()

Initialize the population with random bit vectors.

Get bit vectors for non const populations

```
• bit_vector_t & get_bv (int i)
```

Get a bit vector.

• bit_vector_t & get_best_bv ()

Get best bit vector.

• bit_vector_t & get_best_bv (int i)

Get best bit vector.

• bit_vector_t & get_worst_bv (int i)

Get worst bit vector.

Get bit vectors for const populations

- const bit_vector_t & get_bv (int i) const Get a bit vector.
- const bit_vector_t & get_best_bv () const

Get best bit vector.

const bit_vector_t & get_best_bv (int i) const

Get best bit vector.

• const bit_vector_t & get_worst_bv (int i) const

Get worst bit vector.

Get sorted values

• double get_best_value (int i) const

Get best value.

• double get_best_value () const

Get best value.

Evaluation and sorting

void eval (function::Function *function)

Evaluate the population.

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

Parallel evaluation of the population.

• void sort ()

Sort the lookup table.

void partial_sort (int selection_size)

Partially sort the lookup table.

• void shuffle ()

Shuffle the lookup table.

Selection

• void plus_selection (const Population &offsprings)

Plus selection.

• void plus selection (Population &offsprings)

Plus selection.

• void comma_selection (const Population &offsprings)

Comma selection.

void comma_selection (Population & offsprings)

Comma selection.

Protected Attributes

std::vector< bit_vector_t > _bvs

Bit vectors.

std::vector< index_value_t > _lookup

Lookup table.

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

Binary operator for comparing index-value pairs.

5.85.1 Detailed Description

Population.

Definition at line 36 of file population.hh.

5.85.2 Member Function Documentation

Comma selection.

Implemented with a copy.

Precondition

Offspring population must be partially sorted.

Warning

The function does not break ties randomly (workaround: shuffle offsprings).

Definition at line 112 of file population.cc.

Comma selection.

Implemented with a swap. Should be faster than comma_selection with a copy.

Precondition

Offspring population must be partially sorted.

Warning

The function does not break ties randomly (workaround: shuffle offsprings). Modifies its argument.

Definition at line 126 of file population.cc.

```
5.85.2.3 get_best_bv() [1/4]
bit_vector_t& get_best_bv ( ) [inline]
```

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 85 of file population.hh.

Get best bit vector.

Parameters

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

Precondition

The population must be sorted.

Definition at line 93 of file population.hh.

```
5.85.2.5 get_best_bv() [3/4]
const bit_vector_t& get_best_bv ( ) const [inline]
```

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 117 of file population.hh.

```
5.85.2.6 get_best_bv() [4/4] const bit_vector_t & get_best_bv ( int i ) const [inline]
```

Get best bit vector.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 125 of file population.hh.

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

Get best value.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 148 of file population.hh.

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

Get best value.

Precondition

The population must be sorted.

Definition at line 154 of file population.hh.

```
5.85.2.9 get_worst_bv() [1/2] bit_vector_t@ get_worst_bv ( int i) [inline]
```

Get worst bit vector.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 101 of file population.hh.

```
5.85.2.10 get_worst_bv() [2/2] const bit_vector_t & get_worst_bv ( int <math>i) const [inline]
```

Get worst bit vector.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 133 of file population.hh.

Plus selection.

Implemented with a copy.

Precondition

Both populations must be completely sorted.

Warning

The function does not break ties randomly (workaround: shuffle parents and offsprings).

Definition at line 74 of file population.cc.

Plus selection.

Implemented with a swap. Should be faster than plus_selection with a copy.

Precondition

Both populations must be completely sorted.

Warning

The function does not break ties randomly (workaround: shuffle parents and offsprings). Modifies its argument.

Definition at line 93 of file population.cc.

5.85.3 Member Data Documentation

5.85.3.1 _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.

```
5.85.3.2 _lookup
```

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

Lookup table.

Let p be of type std::pair<int, 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:

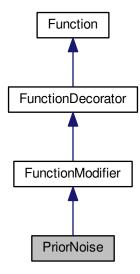
- · lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

5.86 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

• int 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

5.86.1 Detailed Description

Prior noise.

Definition at line 36 of file prior-noise.hh.

5.86.2 Member Function Documentation

```
5.86.2.1 get_maximum()
```

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

Get the global maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 68 of file prior-noise.hh.

5.86.2.2 has_known_maximum()

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

Check for a known maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 74 of file prior-noise.hh.

5.86.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Function.

Definition at line 78 of file prior-noise.hh.

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

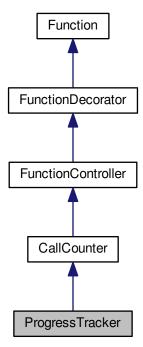
- · lib/hnco/functions/decorators/prior-noise.hh
- · lib/hnco/functions/decorators/prior-noise.cc

5.87 ProgressTracker Class Reference

ProgressTracker.

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

Inheritance diagram for ProgressTracker:



Classes

struct Event

Event.

Public Member Functions

• ProgressTracker (Function *function)

Constructor.

Evaluation

• double eval (const bit_vector_t &)

Evaluate a bit vector.

double incremental_eval (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped
 — bits)

Incremental evaluation.

void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Get information

• const Event & get_last_improvement ()

Get the last improvement.

• double get_evaluation_time ()

Get evaluation time.

Setters

• void set_log_improvement (bool x)

Log improvement.

void set_stream (std::ostream *x)

Output stream.

Protected Member Functions

• void update_last_improvement (double value)

Update last improvement.

Protected Attributes

Event _last_improvement

Last improvement.

• StopWatch _stop_watch

Stop watch.

Parameters

• bool <u>log_improvement</u> = false

Log improvement.

• std::ostream * _stream = &std::cout

Output stream.

5.87.1 Detailed Description

ProgressTracker.

A ProgressTracker is a CallCounter which keeps track the last improvement, that is its value and the number of evaluations needed to reach it.

Definition at line 216 of file function-controller.hh.

5.87.2 Member Function Documentation

5.87.2.1 get_last_improvement()

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

Get the last improvement.

Warning

If _last_improvement.num_evaluations is zero then _function has never been called. The Event returned by get_last_improvement has therefore no meaning.

Definition at line 290 of file function-controller.hh.

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

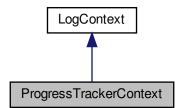
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.88 ProgressTrackerContext Class Reference

Log context for ProgressTracker.

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

Inheritance diagram for ProgressTrackerContext:



Public Member Functions

• ProgressTrackerContext (hnco::function::ProgressTracker *pt)

Constructor.

• std::string get_context ()

Get context.

Private Attributes

hnco::function::ProgressTracker * _pt
 Progress tracker.

5.88.1 Detailed Description

Log context for ProgressTracker.

Definition at line 48 of file log-context.hh.

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

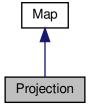
· lib/hnco/algorithms/log-context.hh

5.89 Projection Class Reference

Projection.

#include <hnco/map.hh>

Inheritance diagram for Projection:



Public Member Functions

```
    Projection (const std::vector< int > &bit_positions, int input_size)
    Constructor.
```

void map (const bit_vector_t &input, bit_vector_t &output)

int get_input_size ()

Get input size.

• int get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

• std::vector< int > _bit_positions

Bit positions.

• int _input_size

Input size.

5.89.1 Detailed Description

Projection.

The projection y of a bit vector x is x where we have dropped a given set of components.

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

A projection f from F_2^n to F_2^m , where $n \geq m$, is defined by f(x) = y, where, for all $j \in \{1, 2, \dots, m\}$, $y_j = x_{i_j}$.

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

Definition at line 453 of file map.hh.

5.89.2 Constructor & Destructor Documentation

5.89.2.1 Projection()

Constructor.

The output size of the map is given by the size of bit_positions.

Parameters

bit_positions	Bit positions in the input from where output bits are copied
input_size	Input size

Precondition

```
input_size >= bit_positions.size()
```

Definition at line 164 of file map.cc.

5.89.3 Member Function Documentation

5.89.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 491 of file map.hh.

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

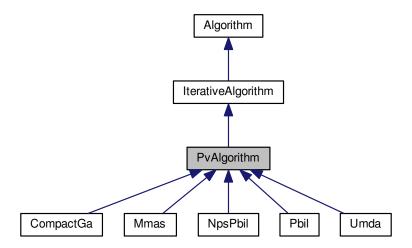
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.90 PvAlgorithm Class Reference

Probability vector algorithm.

#include <hnco/algorithms/pv/pv-algorithm.hh>

Inheritance diagram for PvAlgorithm:



Public Member Functions

• PvAlgorithm (int n)

Constructor.

Setters for logging

• void set_log_entropy (bool x)

Log entropy.

• void set_log_num_components (int x)

Set the number of probability vector components to log.

void set_log_pv (bool x)

Log probability vector.

• void set_something_to_log ()

Set flag for something 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.

Logging

• bool <u>log_entropy</u> = false Log entropy.

bool <u>log_pv</u> = false

Log probability vector.

• int _log_num_components = 5

Number of probability vector components to log.

5.90.1 Detailed Description

Probability vector algorithm.

Definition at line 35 of file pv-algorithm.hh.

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

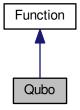
- lib/hnco/algorithms/pv/pv-algorithm.hh
- · lib/hnco/algorithms/pv/pv-algorithm.cc

5.91 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.

• int 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.

5.91.1 Detailed Description

Quadratic unconstrained binary optimization.

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

5.91.2 Member Function Documentation

```
5.91.2.1 load()
```

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

Load an instance.

Exceptions

Error

Definition at line 35 of file qubo.cc.

5.91.3 Member Data Documentation

```
5.91.3.1 _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

5.92 Random Struct Reference

Random numbers.

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

Static Public Member Functions

• static double uniform ()

Next uniformly distributed sample.

• static double normal ()

Next normally distributed sample.

• static bool bernoulli ()

Next random bit.

Static Public Attributes

• static std::mt19937 generator

Mersenne Twister 19937 generator.

5.92.1 Detailed Description

Random numbers.

Definition at line 33 of file random.hh.

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

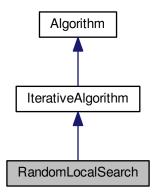
- · lib/hnco/random.hh
- · lib/hnco/random.cc

5.93 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.

5.93.1 Detailed Description

Random local search.

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

5.93.2 Member Function Documentation

5.93.2.1 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:

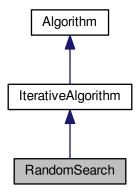
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

5.94 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

5.94.1 Detailed Description

Random search.

Definition at line 31 of file random-search.hh.

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

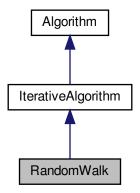
- lib/hnco/algorithms/random-search.hh
- · lib/hnco/algorithms/random-search.cc

5.95 RandomWalk Class Reference

Random walk.

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

Inheritance diagram for RandomWalk:



Public Member Functions

RandomWalk (int n, neighborhood::Neighborhood *neighborhood)

Constructor.

· void init ()

Random initialization.

void init (const bit_vector_t &x)

Explicit initialization.

void init (const bit_vector_t &x, double value)

Explicit initialization.

• void log ()

Log.

Setters

• void set_incremental_evaluation (bool x)

Set incremental evaluation.

 void set_log_value () Set log.

Protected Member Functions

· void iterate ()

Single iteration.

· void iterate_full ()

Single iteration with full evaluation.

void iterate_incremental ()

Single iteration with incremental evaluation.

Protected Attributes

neighborhood::Neighborhood * neighborhood

Neighborhood.

· double _value

Value of the last visited bit vector.

Parameters

 bool _incremental_evaluation = false Incremental evaluation.

5.95.1 Detailed Description

Random walk.

The algorithm simply performs a random walk on the graph implicitly given by the neighborhood. At each iteration, the chosen neighbor does not depend on its evaluation. However optimization takes place as in random search, that is the best visited bit vector is remembered.

Definition at line 42 of file random-walk.hh.

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

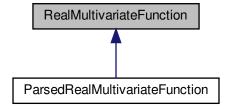
- lib/hnco/algorithms/ls/random-walk.hh
- lib/hnco/algorithms/ls/random-walk.cc

5.96 RealMultivariateFunction Class Reference

Real multivariate function.

#include <hnco/functions/real/real-multivariate-function.hh>

Inheritance diagram for RealMultivariateFunction:



Public Member Functions

virtual ∼RealMultivariateFunction ()

Destructor.

• virtual int get_dimension ()=0

Get the dimension of vectors.

• virtual double eval (const std::vector< double > x)=0

Evaluate a real vector.

5.96.1 Detailed Description

Real multivariate function.

Definition at line 35 of file real-multivariate-function.hh.

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

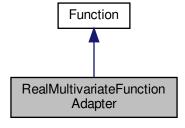
· lib/hnco/functions/real/real-multivariate-function.hh

5.97 RealMultivariateFunctionAdapter Class Reference

Real multivariate function adapter.

#include <hnco/functions/real/real-multivariate-function-adapter.hh>

Inheritance diagram for RealMultivariateFunctionAdapter:



Public Member Functions

RealMultivariateFunctionAdapter (RealRepresentation *rep, RealMultivariateFunction *fn)
 Constructor.

Information about the function

```
• int get_bv_size ()

Get bit vector size.
```

Evaluation

double eval (const bit_vector_t &x)
 Evaluate a bit vector.

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.

Private Attributes

• RealRepresentation * _representation

Real representation.

• RealMultivariateFunction * _function

Real multivariate function.

std::vector< double > _rv
 Real vector.

5.97.1 Detailed Description

Real multivariate function adapter.

Definition at line 37 of file real-multivariate-function-adapter.hh.

5.97.2 Constructor & Destructor Documentation

5.97.2.1 RealMultivariateFunctionAdapter()

Constructor.

Parameters

rep	Real representation
fn	Real multivariate function

Definition at line 59 of file real-multivariate-function-adapter.hh.

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

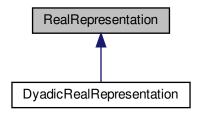
- lib/hnco/functions/real/real-multivariate-function-adapter.hh
- lib/hnco/functions/real/real-multivariate-function-adapter.cc

5.98 RealRepresentation Class Reference

Real representation.

#include <hnco/functions/real/real-representation.hh>

Inheritance diagram for RealRepresentation:



Public Member Functions

- virtual ∼RealRepresentation ()
 - Destructor.
- virtual int size ()=0
 - Size of the representation.
- virtual double convert (hnco::bit_vector_t::const_iterator first, hnco::bit_vector_t::const_iterator last)=0

 Convert a bit vector range into a double.

5.98.1 Detailed Description

Real representation.

Definition at line 35 of file real-representation.hh.

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

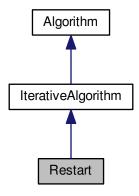
• lib/hnco/functions/real/real-representation.hh

5.99 Restart Class Reference

Restart.

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

Inheritance diagram for Restart:



Public Member Functions

- Restart (int n, Algorithm *algorithm)
 - Constructor.
- void init ()

Initialization.

• void set_function (function::Function *function)

Set function

void set_functions (const std::vector< function::Function *> functions)
 Set functions.

Private Member Functions

• void iterate ()

Optimize.

Private Attributes

Algorithm * _algorithm
 Algorithm.

Additional Inherited Members

5.99.1 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:

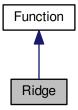
- lib/hnco/algorithms/decorators/restart.hh
- lib/hnco/algorithms/decorators/restart.cc

5.100 Ridge Class Reference

Ridge.

#include <hnco/functions/theory.hh>

Inheritance diagram for Ridge:



Public Member Functions

• Ridge (int bv_size)

Constructor.

• int 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

• int _bv_size

Bit vector size.

5.100.1 Detailed Description

Ridge.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 208 of file theory.hh.

5.100.2 Member Function Documentation

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

Get the global maximum.

Returns

```
2 * _bv_size
```

Reimplemented from Function.

Definition at line 232 of file theory.hh.

```
5.100.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 228 of file theory.hh.

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

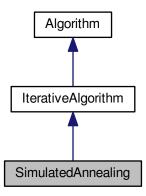
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.101 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

5.101.1 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.

5.101.2 Member Function Documentation

```
5.101.2.1 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:

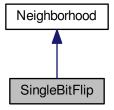
- lib/hnco/algorithms/ls/simulated-annealing.hh
- lib/hnco/algorithms/ls/simulated-annealing.cc

5.102 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

5.102.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

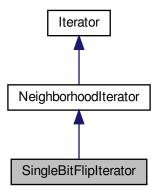
• lib/hnco/neighborhoods/neighborhood.hh

5.103 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

5.103.1 Detailed Description

Single bit flip neighborhood iterator.

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

5.103.2 Constructor & Destructor Documentation

5.103.2.1 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:

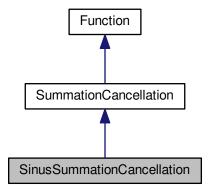
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.104 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

5.104.1 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 104 of file cancellation.hh.

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

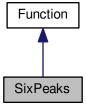
- · lib/hnco/functions/cancellation.hh
- · lib/hnco/functions/cancellation.cc

5.105 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



Public Member Functions

· SixPeaks (int by size, int threshold)

Constructor.

• int 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

• int _bv_size

Bit vector size.

· int _threshold

Threshold.

int maximum

Maximum.

5.105.1 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.

5.105.2 Member Function Documentation

```
5.105.2.1 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.

```
5.105.2.2 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

5.106 SpinHerding Class Reference

Herding with spin variables.

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

Public Types

• enum { SAMPLE_GREEDY, SAMPLE_RLS, SAMPLE_DLS, LAST_SAMPLE }

Public Member Functions

• 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.

Getters

const SpinMoment & get_delta ()
 Get delta.

Setters

void set_randomize_bit_order (bool x)

Randomize bit order.

void set_sampling_method (int x)

Set the sampling method.

void set_num_seq_updates (int x)

Set the number of sequential updates per sample.

• void set_weight (double x)

Set the weight of second order moments.

Protected Member Functions

void compute_delta (const SpinMoment &target)

Compute delta.

void sample_greedy (bit_vector_t &x)

Sample by means of a greedy algorithm.

• double q_derivative (const bit_vector_t &x, int i)

Derivative of q.

• double q_variation (const bit_vector_t &x, int i)

Variation of q.

void sample_rls (bit_vector_t &x)

Sample by means of random local search.

void sample_dls (bit_vector_t &x)

Sample by means of deterministic local search.

Protected Attributes

• SpinMoment _delta

Delta moment.

SpinMoment _count

Counter moment.

• permutation_t _permutation

Permutation.

 $\bullet \quad \mathsf{std} :: \mathsf{uniform_int_distribution} < \mathsf{int} > _\mathsf{choose_bit}$

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.

• double _weight = 1

Weight of second order moments.

5.106.1 Detailed Description

Herding with spin variables.

By spin variables, we mean variables taking values 1 or -1, instead of 0 or 1 in the case of binary variables.

Definition at line 37 of file spin-herding.hh.

5.106.2 Member Enumeration Documentation

5.106.2.1 anonymous enum

anonymous enum

Enumerator

SAMPLE_GREEDY	Greedy algorithm.
SAMPLE_RLS	Random local search.
SAMPLE_DLS	Deterministic local search.

Definition at line 97 of file spin-herding.hh.

5.106.3 Constructor & Destructor Documentation

5.106.3.1 SpinHerding()

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

Constructor.

Parameters

```
n Size of bit vectors
```

_num_seq_updates is initialized to n.

Definition at line 116 of file spin-herding.hh.

5.106.4 Member Function Documentation

5.106.4.1 q_variation()

```
double q_variation (  {\rm const\ bit\_vector\_t\ \&\ x,}  int i ) [protected]
```

Variation of q.

Up to a positive multiplicative constant. Only the sign of the variation matters to local search.

Definition at line 162 of file spin-herding.cc.

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

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

5.107 SpinMoment Struct Reference

Moment for spin variables.

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

Public Member Functions

• SpinMoment (int n)

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

• void init ()

Initialize accumulators.

void add (const bit_vector_t &x)

Update accumulators.

· void average (int count)

Compute average.

void update (const SpinMoment &p, double rate)

Update moment.

· void bound (double margin)

Bound moment.

• double distance (const SpinMoment &p) const

Distance.

• double norm_2 () const

Compute the norm 2.

• double diameter () const

Compute the diameter.

• size_t size () const

Size.

· void display (std::ostream &stream)

Display.

Public Attributes

• std::vector< double > _first

First moment.

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

Second moment.

• double _weight = 1

Weight of second order moments.

5.107.1 Detailed Description

Moment for spin variables.

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

5.107.2 Member Data Documentation

5.107.2.1 _second

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

Second moment.

This is a lower triangular matrix with only zeros on the diagonal. Only entries $_$ second[i][j] with j < i are considered.

Definition at line 50 of file spin-moment.hh.

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

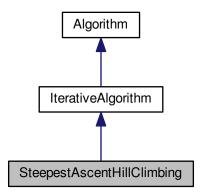
- lib/hnco/algorithms/hea/spin-moment.hh
- lib/hnco/algorithms/hea/spin-moment.cc

5.108 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::Neighborhood)terator *neighborhood)
 - Constructor.
- void init ()

Random initialization.

- void init (const bit_vector_t &x)
 - Explicit initialization.
- void init (const bit_vector_t &x, double value)

Explicit initialization.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

std::vector < bit_vector_t > _candidates
 Potential candidate.

neighborhood::NeighborhoodIterator * _neighborhood
 Neighborhood.

5.108.1 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:

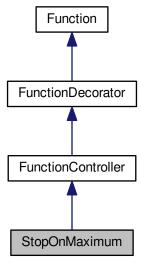
- · lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

5.109 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

5.109.1 Detailed Description

Stop on maximum.

The member function eval throws an exception MaximumReached when its argument maximizes the decorated function.

Warning

The maximum is detected using the equality operator hence the result should be taken with care in case of non integer (floating point) function values.

Definition at line 90 of file function-controller.hh.

5.109.2 Constructor & Destructor Documentation

5.109.2.1 StopOnMaximum()

Constructor.

Parameters

function Decorated function

Precondition

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

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

5.109.3 Member Function Documentation

Evaluate a bit vector.

Exceptions

MaximumReached

Implements Function.

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

5.109.3.2 incremental_eval()

Incremental evaluation.

Exceptions

```
MaximumReached
```

Reimplemented from Function.

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

5.109.3.3 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:

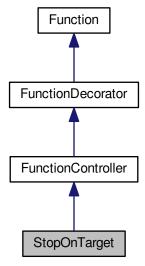
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.110 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

5.110.1 Detailed Description

Stop on target.

The member function eval throws an exception TargetReached when the value of its decorated function reaches a given target.

Warning

The target is detected using the greater or equal operator hence the result should be taken with care in case of non integer (floating point) function values.

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

5.110.2 Constructor & Destructor Documentation

5.110.2.1 StopOnTarget()

Constructor.

Parameters

function	Decorated function
target	Target

Definition at line 147 of file function-controller.hh.

5.110.3 Member Function Documentation

Evaluate a bit vector.

Exceptions

TargetReached

Implements Function.

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

5.110.3.2 incremental_eval()

Incremental evaluation.

Exceptions

TargetReached

Reimplemented from Function.

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

5.110.3.3 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

5.111 StopWatch Class Reference

Stop watch.

```
#include <hnco/stop-watch.hh>
```

Public Member Functions

```
• void start ()
```

Start.

• void stop ()

Stop.

• double get_total_time ()

Get total time.

Private Attributes

```
• double _total_time = 0
```

Total time.

clock_t <u>start</u>

Start time.

5.111.1 Detailed Description

Stop watch.

Definition at line 31 of file stop-watch.hh.

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

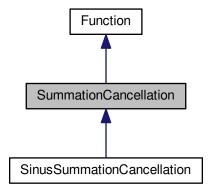
· lib/hnco/stop-watch.hh

5.112 SummationCancellation Class Reference

Summation cancellation.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SummationCancellation:



Public Member Functions

• SummationCancellation (int n)

Constructor.

int 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

```
    int _bv_size
        Bit vector size.
    std::vector< double > _buffer
        Buffer.
```

5.112.1 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 48 of file cancellation.hh.

5.112.2 Constructor & Destructor Documentation

5.112.2.1 SummationCancellation()

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

Constructor.

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

Parameters

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

Definition at line 71 of file cancellation.hh.

5.112.3 Member Function Documentation

5.112.3.1 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 cancellation.hh.

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

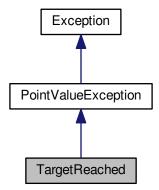
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.113 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

5.113.1 Detailed Description

target reached

Definition at line 62 of file exception.hh.

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

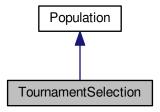
· lib/hnco/exception.hh

5.114 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

5.114.1 Detailed Description

Population with tournament selection.

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

5.114.2 Member Function Documentation

```
5.114.2.1 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:

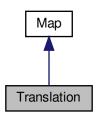
- · lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

5.115 Translation Class Reference

Translation.

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

Inheritance diagram for Translation:



Public Member Functions

• void random (int n)

Random instance.

• void map (const bit_vector_t &input, bit_vector_t &output)

Мар

• int get_input_size ()

Get input size.

• int 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

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

5.115.1 Detailed Description

Translation.

A translation is an affine map f from F_2y^n to itself defined by f(x)=x+b, where b is an n-dimensional bit vector.

Definition at line 71 of file map.hh.

5.115.2 Member Function Documentation

```
5.115.2.1 is_surjective()
```

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 122 of file map.hh.

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

- · lib/hnco/map.hh
- lib/hnco/map.cc

5.116 Transvection Struct Reference

Transvection.

#include <hnco/transvection.hh>

Public Member Functions

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

• template < class Archive >

void load (Archive &ar, const unsigned int version)

Load.

• bool is_valid () const

Check validity.

bool is_valid (int n) const

Check validity.

• void display (std::ostream &stream) const

Display transvection.

• void random (int n)

Sample a random transvection.

• void random_non_commuting (int n, const Transvection &a)

Sample a random transvection.

void multiply (bit_vector_t &x) const

Multiply a bit vector from the left.

void multiply (bit_matrix_t &M) const

Multiply a bit matrix from the left.

void multiply_right (bit_matrix_t &M) const

Multiply a bit matrix from the right.

Public Attributes

int row_index

Row index.

• int column_index

Column index.

5.116.1 Detailed Description

Transvection.

We only consider transvections defined by matrices $\tau_{ij} = I_n + B_{ij}$, where I_n is the $n \times n$ identity matrix and B_{ij} is the matrix whose (i,j) entry is 1 and other entries are zero. Such a matrix is also sometimes called a shear matrix.

Transvections generate invertible matrices over the finite field F_2 .

Definition at line 53 of file transvection.hh.

5.116.2 Member Function Documentation

5.116.2.1 is_valid()

Check validity.

Parameters

```
n Dimension
```

Definition at line 46 of file transvection.cc.

Multiply a bit vector from the left.

Parameters

```
x Bit vector
```

Precondition

```
is_valid()
is_valid(x.size())
```

Warning

This function modifies the given bit vector.

Definition at line 102 of file transvection.cc.

Multiply a bit matrix from the left.

Parameters

```
M Bit matrix
```

Precondition

```
is_valid()
is_valid(bm_num_rows(M))
```

Warning

This function modifies the given bit vector.

Definition at line 114 of file transvection.cc.

5.116.2.4 multiply_right()

Multiply a bit matrix from the right.

Parameters

```
M Bit matrix
```

Precondition

```
is_valid()
is_valid(bm_num_rows(M))
```

Warning

This function modifies the given bit vector.

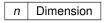
Definition at line 124 of file transvection.cc.

5.116.2.5 random()

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

Sample a random transvection.

Parameters



Precondition

n > 1

Definition at line 59 of file transvection.cc.

5.116.2.6 random_non_commuting()

```
void random_non_commuting (  \qquad \qquad \text{int } n, \\  \qquad \qquad \text{const Transvection \& } a \text{ )}
```

Sample a random transvection.

This member function ensures that the sampled transvection does not commute with some given one.

Parameters

n	Dimension
а	Given transvection

Precondition

n > 1

Definition at line 75 of file transvection.cc.

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

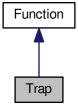
- · lib/hnco/transvection.hh
- lib/hnco/transvection.cc

5.117 Trap Class Reference

Trap.

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

Inheritance diagram for Trap:



Public Member Functions

```
• Trap (int bv_size, int num_traps)
```

Constructor.

• int 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

• int _bv_size

Bit vector size.

• int _num_traps

Number of traps.

int _trap_size

Trap size.

5.117.1 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.

5.117.2 Constructor & Destructor Documentation

```
5.117.2.1 Trap()
```

Constructor.

Parameters

bv_size	Bit vector size	
num_traps	Number of traps	

Warning

bv_size must be a multiple of num_traps

Definition at line 64 of file trap.hh.

5.117.3 Member Function Documentation

```
5.117.3.1 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.

5.117.3.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 84 of file trap.hh.

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

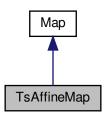
- · lib/hnco/functions/trap.hh
- lib/hnco/functions/trap.cc

5.118 TsAffineMap Class Reference

Transvection sequence affine map.

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

Inheritance diagram for TsAffineMap:



Public Types

enum SamplingMode {
 Unconstrained, CommutingTransvections, UniqueSource, UniqueDestination,
 DisjointTransvections, NonCommutingTransvections }

 Sampling mode.

Public Member Functions

• void random (int n, int t, SamplingMode mode)

Random instance.

void map (const bit_vector_t &input, bit_vector_t &output)

Мар

• int get_input_size ()

Get input size.

• int get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

• transvection_sequence_t _ts

Transvection sequence

bit_vector_t _bv

Translation vector

Friends

· class boost::serialization::access

5.118.1 Detailed Description

Transvection sequence affine map.

An affine map f from F_2^m to F_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.

In TsAffineMap, A is a finite product of transvections represented by a transvection_sequence_t.

Definition at line 505 of file map.hh.

5.118.2 Member Enumeration Documentation

5.118.2.1 SamplingMode

enum SamplingMode

Sampling mode.

Enumerator

Unconstrained	Unconstrained.
CommutingTransvections	Commuting transvections.
UniqueSource	Transvection sequence with unique source
UniqueDestination	Transvection sequence with unique destination
DisjointTransvections	Disjoint transvections.
NonCommutingTransvections	Non commuting transvections.

Definition at line 542 of file map.hh.

5.118.3 Member Function Documentation

```
5.118.3.1 is_surjective()
```

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 585 of file map.hh.

5.118.3.2 random()

Random instance.

Parameters

n	Dimension
t	Length of sequence of transvections
mode	Sampling mode

Definition at line 185 of file map.cc.

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

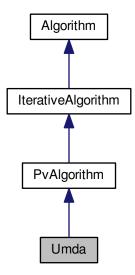
- · lib/hnco/map.hh
- lib/hnco/map.cc

5.119 Umda Class Reference

Univariate marginal distribution algorithm.

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

Inheritance diagram for Umda:



Public Member Functions

- Umda (int n, int population_size)

 Constructor.
- void init ()

 Initialization.

Setters

• void set_selection_size (int x)

Set the selection size.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

• Population _population Population.

Parameters

• int _selection_size = 1 Selection size.

5.119.1 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 41 of file umda.hh.

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

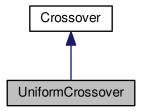
- lib/hnco/algorithms/pv/umda.hh
- · lib/hnco/algorithms/pv/umda.cc

5.120 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.

5.120.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

5.120.2 Member Function Documentation

5.120.2.1 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:

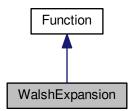
- · lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

5.121 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



Public Member Functions

• WalshExpansion ()

Constructor.

int get_bv_size ()

Get bit vector size.

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

Instance generators

template < class Generator > void generate (int n, int num_features, Generator generator)

Instance generator.

• void random (int n, int num_features)

Random instance.

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

5.121.1 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.

5.121.2 Member Function Documentation

5.121.2.1 generate()

```
void generate (
          int n,
          int num_features,
          Generator generator ) [inline]
```

Instance generator.

Parameters

n	Size of bit vectors
num_features	Number of feature vectors
generator	Coefficient generator

Definition at line 87 of file walsh-expansion.hh.

5.121.2.2 random()

Random instance.

The coefficients are sampled from the normal distribution.

Parameters

n	Size of bit vector
num_features	Number of feature vectors

Definition at line 113 of file walsh-expansion.hh.

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

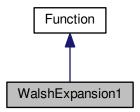
- · lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

5.122 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.

Instance generators

template < class Generator > void generate (int n, Generator generator)

Instance generator.

void random (int n)

Random instance.

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.

Information about the function

• int 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.

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

5.122.1 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.

5.122.2 Member Function Documentation

5.122.2.1 generate()

```
void generate (  \mbox{int } n, \\ \mbox{Generator } generator \mbox{)} \mbox{ [inline]}
```

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

Definition at line 83 of file walsh-expansion-1.hh.

5.122.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 130 of file walsh-expansion-1.hh.

5.122.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 135 of file walsh-expansion-1.hh.

5.122.2.4 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

```
n Size of bit vectors
```

Definition at line 97 of file walsh-expansion-1.hh.

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

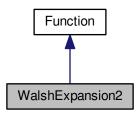
- lib/hnco/functions/walsh/walsh-expansion-1.hh
- lib/hnco/functions/walsh/walsh-expansion-1.cc

5.123 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.

• int get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Instance generators

template < class LinearGen , class QuadraticGen > void generate (int n, LinearGen linear_gen, QuadraticGen quadratic_gen)

Instance generators.

void random (int n)

Instance generator.

• void generate_ising1_long_range (int n, double alpha)

Generate one dimensional Ising model with long range interactions.

• void generate_ising1_long_range_periodic (int n, double alpha)

Generate one dimensional Ising model with long range interactions and periodic boundary conditions.

Private Member Functions

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

• void resize (int n)

Resize data structures.

Private Attributes

• std::vector< double > _linear

Linear part.

Quadratic part.

Friends

· class boost::serialization::access

5.123.1 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

$$f(x) = \sum_{i} a_i (-1)^{x_i} + \sum_{i < j} a_{ij} (-1)^{x_i + x_j}$$

Definition at line 50 of file walsh-expansion-2.hh.

5.123.2 Member Function Documentation

5.123.2.1 generate()

```
void generate (
          int n,
          LinearGen linear_gen,
          QuadraticGen quadratic_gen ) [inline]
```

Instance generators.

Parameters

n	Size of bit vectors
linear_gen	Generator for the linear part
quadratic_gen	Generator for the quadratic part

Definition at line 95 of file walsh-expansion-2.hh.

5.123.2.2 generate_ising1_long_range()

```
void generate_ising1_long_range (
          int n,
           double alpha )
```

Generate one dimensional Ising model with long range interactions.

Similar to a Dyson-Ising model except for the finite, instead of infinite, linear chain of spins.

Its expression is of the form

$$f(x) = \sum_{ij} J(d_{ij})(1 - 2x_i)(1 - 2x_j)$$

or equivalently

$$f(x) = \sum_{ij} J(d_{ij})(-1)^{x_i + x_j}$$

where $J(d_{ij})$ is the interaction between sites i and j, $d_{ij}=|i-j|$, and $J(n)=n^{-\alpha}$.

Since we are maximizing f or minimizing -f, the expression of f is compatible with what can be found in physics textbooks.

Parameters

n	Size of bit vectors
alpha	Exponential decay parameter

Definition at line 82 of file walsh-expansion-2.cc.

5.123.2.3 generate_ising1_long_range_periodic()

Generate one dimensional Ising model with long range interactions and periodic boundary conditions.

Similar to a Dyson-Ising model except for the finite, instead of infinite, linear chain of spins.

Its expression is of the form

$$f(x) = \sum_{i,j} J(d_{ij})(1 - 2x_i)(1 - 2x_j)$$

or equivalently

$$f(x) = \sum_{ij} J(d_{ij})(-1)^{x_i + x_j}$$

where $J(d_{ij})$ is the interaction between sites i and j, $d_{ij} = \min\{|i-j|, n-|i-j|\}$, and $J(n) = n^{-\alpha}$.

Since we are maximizing f or minimizing -f, the expression of f is compatible with what can be found in physics textbooks.

Parameters

n	Size of bit vectors
alpha	Exponential decay parameter

Definition at line 103 of file walsh-expansion-2.cc.

5.123.2.4 random()

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

Instance generator.

The weights are sampled from the normal distribution.

Parameters

```
n Size of bit vector
```

Definition at line 117 of file walsh-expansion-2.hh.

5.123.3 Member Data Documentation

5.123.3.1 _quadratic

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

Quadratic part.

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

Definition at line 73 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

5.124 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)

Public Attributes

• std::vector< bool > feature

Feature.

· double coefficient

Coefficient.

5.124.1 Detailed Description

Walsh transform term.

Definition at line 46 of file function.hh.

5.124.2 Member Data Documentation

5.124.2.1 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 53 of file function.hh.

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

• lib/hnco/functions/function.hh

Index

_compare_index_value	hnco::algorithm::BiasedCrossover, 51
hnco::algorithm::Population, 198	hnco::algorithm::Crossover, 66
_expression	hnco::algorithm::UniformCrossover, 260
hnco::function::AbstractMaxSat, 41	bv_from_vector_bool
_functions	hnco, 23
hnco::algorithm::Algorithm, 48	bv_to_vector_bool
_lookup	hnco, 23
hnco::algorithm::Population, 198	
_q	Cache, 59
hnco::function::Qubo, 210	hnco::function::Cache, 60
_quadratic	CallCounter, 61
hnco::function::WalshExpansion2, 269	comma_selection
_second	hnco::algorithm::Population, 194
hnco::algorithm::hea::SpinMoment, 233	CompactGa, 62
	CompleteSearch, 64
AbstractLabs, 39	compute_walsh_transform
AbstractMaxSat, 40	hnco::function::Function, 83
AdditiveGaussianNoise, 42	Crossover, 65
AffineMap, 43	
Algorithm, 45	DeceptiveJump, 66
- ngo,	DyadicRealRepresentation, 68
bernoulli trials	hnco::function::real::DyadicRealRepresentation, 69
hnco::neighborhood::MultiBitFlip, 149	
BernoulliProcess, 48	EqualProducts, 70
hnco::neighborhood::BernoulliProcess, 49, 50	Error, 73
BiasedCrossover, 51	eval
bit_t	hnco::function::NearestNeighborIsingModel1, 155
hnco, 18	hnco::function::NearestNeighborIsingModel2, 158
BitHerding, 52	hnco::function::OnBudgetFunction, 173
BitMoment, 54	hnco::function::StopOnMaximum, 237
bm_add_columns	hnco::function::StopOnTarget, 240
hnco, 19	Exception, 74
bm_add_rows	Factorization, 75
hnco, 19	hnco::function::Factorization, 76
bm_identity	feature
hnco, 20	hnco::function::Function::WalshTransformTerm,
bm_invert	270
hnco, 20	FirstAscentHillClimbing, 77
bm_multiply	FourPeaks, 79
hnco, 21	Function, 81
bm_rank	Function::WalshTransformTerm, 269
hnco, 21	FunctionController, 86
bm_row_echelon_form	FunctionDecorator, 87
hnco, 21	FunctionMapComposition, 88
bm_solve	hnco::function::FunctionMapComposition, 89
hnco, 22	FunctionModifier, 91
bm_solve_upper_triangular	FunctionPlugin, 92
hnco, 22	hnco::function::FunctionPlugin, 93
BmPbil, 55	• .
breed	generate

hnco::function::EqualProducts, 71	hnco::function::SixPeaks, 229
hnco::function::LinearFunction, 124	hnco::function::SummationCancellation, 24
hnco::function::NearestNeighborIsingModel1, 155	hnco::function::Trap, 254
hnco::function::NearestNeighborIsingModel2, 158	hnco::function::WalshExpansion1, 264
hnco::function::NkLandscape, 169	Hboa, 101
hnco::function::WalshExpansion, 262	Hea
hnco::function::WalshExpansion1, 264	hnco::algorithm::hea::Hea, 105
hnco::function::WalshExpansion2, 267	Hea< Moment, Herding >, 102
generate_ising1_long_range	Hiff, 107
hnco::function::WalshExpansion2, 267	hnco, 13
generate_ising1_long_range_periodic	bit_t, 18
hnco::function::WalshExpansion2, 268	bm_add_columns, 19
GeneticAlgorithm, 93	bm_add_rows, 19
hnco::algorithm::GeneticAlgorithm, 95	bm_identity, 20
get_best_bv	bm_invert, 20
hnco::algorithm::Population, 194, 195	bm_multiply, 21
get_best_value	bm_rank, 21
hnco::algorithm::Population, 196	bm_row_echelon_form, 21
get_last_improvement	bm solve, 22
hnco::function::ProgressTracker, 203	bm_solve_upper_triangular, 22
get_maximum	bv_from_vector_bool, 23
hnco::function::DeceptiveJump, 67	bv_to_vector_bool, 23
hnco::function::FourPeaks, 80	is_in_range, 24
hnco::function::Function, 84	perm_identity, 25
hnco::function::FunctionMapComposition, 90	perm_random, 25
hnco::function::Hiff, 108	sbm_multiply, 25
hnco::function::Jump, 117	sparse_bit_matrix_t, 18
hnco::function::LeadingOnes, 121	sparse_bit_vector_t, 18
hnco::function::LongPath, 130	transvection_sequence_t, 18
hnco::function::Needle, 161	ts_is_valid, 25, 26
hnco::function::OneMax, 176	
hnco::function::Plateau, 190	ts_multiply, 26
hnco::function::PriorNoise, 200	ts_random, 27
hnco::function::Ridge, 221	ts_random_commuting, 27
hnco::function::SixPeaks, 229	ts_random_disjoint, 28
hnco::function::Trap, 254	ts_random_non_commuting, 28
get_worst_bv	ts_random_unique_destination, 29
hnco::algorithm::Population, 196, 197	ts_random_unique_source, 29
	hnco::AffineMap
HammingBall, 96	is_surjective, 44
hnco::neighborhood::HammingBall, 97	random, 44
HammingSphere, 98	hnco::Injection
hnco::neighborhood::HammingSphere, 99	Injection, 111
HammingSphereIterator, 99	hnco::LinearMap
hnco::neighborhood::HammingSphereIterator, 101	is_surjective, 126
has_known_maximum	random, 126
hnco::function::DeceptiveJump, 67	hnco::Map
hnco::function::FourPeaks, 80	is_surjective, 134
hnco::function::FunctionMapComposition, 90	hnco::MapComposition
hnco::function::Hiff, 108	is_surjective, 136
hnco::function::Jump, 117	MapComposition, 135
hnco::function::LeadingOnes, 121	hnco::Permutation
hnco::function::LinearFunction, 124	is_surjective, 188
hnco::function::LongPath, 131	hnco::Projection
hnco::function::Needle, 161	is_surjective, 206
hnco::function::OneMax, 176	Projection, 205
hnco::function::Plateau, 190	hnco::Translation
hnco::function::PriorNoise, 200	is_surjective, 248
hnco::function::Ridge, 221	hnco::Transvection

is_valid, 249	Hea, 105
multiply, 250	set_reset_period, 106
multiply_right, 251	set_selection_size, 106
random, 251	hnco::algorithm::hea::SpinHerding
random non commuting, 251	q variation, 232
hnco::TsAffineMap	SpinHerding, 232
is_surjective, 256	hnco::algorithm::hea::SpinMoment
random, 257	second, 233
SamplingMode, 256	hnco::exception, 33
hnco::algorithm, 30	hnco::function, 34
hnco::algorithm::Algorithm	hnco::function::AbstractMaxSat
_functions, 48	_expression, 41
set_solution, 47	load, 41
update_solution, 47	hnco::function::Cache
hnco::algorithm::BiasedCrossover	Cache, 60
breed, 51	provides_incremental_evaluation, 60
hnco::algorithm::Crossover	hnco::function::DeceptiveJump
breed, 66	get maximum, 67
hnco::algorithm::GeneticAlgorithm	has_known_maximum, 67
GeneticAlgorithm, 95	hnco::function::EqualProducts
set allow stay, 96	generate, 71
hnco::algorithm::IterativeAlgorithm	random, 71
IterativeAlgorithm, 113	hnco::function::Factorization
maximize, 114	Factorization, 76
set_num_iterations, 114	load, 77
hnco::algorithm::MuCommaLambdaEa	hnco::function::FourPeaks
MuCommaLambdaEa, 147	get_maximum, 80
set_allow_stay, 148	has_known_maximum, 80
hnco::algorithm::MuPlusLambdaEa	hnco::function::Function
MuPlusLambdaEa, 152	compute_walsh_transform, 83
set_allow_stay, 152	get_maximum, 84
hnco::algorithm::OnePlusLambdaCommaLambdaGa	incremental_eval, 84
OnePlusLambdaCommaLambdaGa, 178	provides incremental evaluation, 85
hnco::algorithm::OnePlusOneEa	safe eval, 85
OnePlusOneEa, 180	hnco::function::Function::WalshTransformTerm
set_allow_stay, 181	feature, 270
set_num_iterations, 181	hnco::function::FunctionController
hnco::algorithm::Population	provides incremental evaluation, 87
compare index value, 198	hnco::function::FunctionMapComposition
lookup, 198	FunctionMapComposition, 89
comma_selection, 194	get_maximum, 90
get_best_bv, 194, 195	has_known_maximum, 90
get_best_value, 196	hnco::function::FunctionPlugin
get worst bv, 196, 197	FunctionPlugin, 93
plus_selection, 197	hnco::function::Hiff
hnco::algorithm::RandomLocalSearch	get_maximum, 108
set_patience, 212	has_known_maximum, 108
hnco::algorithm::SimulatedAnnealing	hnco::function::Jump
init_beta, 223	get_maximum, 117
hnco::algorithm::TournamentSelection	has_known_maximum, 117
select, 246	hnco::function::LeadingOnes
hnco::algorithm::UniformCrossover	get_maximum, 121
breed, 260	has_known_maximum, 121
hnco::algorithm::bm_pbil, 33	hnco::function::LinearFunction
hnco::algorithm::bm_pbil::BmPbil	generate, 124
set_selection_size, 58	has_known_maximum, 124
hnco::algorithm::hea, 33	provides_incremental_evaluation, 124
hnco::algorithm::hea::Hea	random, 124

hnco::function::LongPath	update, 237
get_maximum, 130	hnco::function::StopOnTarget
has_known_maximum, 131	eval, 240
hnco::function::MaxNae3Sat	incremental_eval, 240
load, 138	StopOnTarget, 239
hnco::function::MaxSat	update, 240
random, 139, 140	hnco::function::SummationCancellation
hnco::function::NearestNeighborIsingModel1	has_known_maximum, 244
eval, 155	SummationCancellation, 243
generate, 155	hnco::function::Trap
provides_incremental_evaluation, 155	get_maximum, 254
random, 155	has_known_maximum, 254
hnco::function::NearestNeighborIsingModel2	Trap, 253
eval, 158	hnco::function::WalshExpansion
generate, 158	generate, 262
provides_incremental_evaluation, 159	random, 262
random, 159	hnco::function::WalshExpansion1
hnco::function::Needle	generate, 264
get_maximum, 161	has_known_maximum, 264
has_known_maximum, 161	provides_incremental_evaluation, 265
hnco::function::Negation	random, 265
provides_incremental_evaluation, 163	hnco::function::WalshExpansion2
hnco::function::NkLandscape	_quadratic, 269
generate, 169	generate, 267
random, 170	generate_ising1_long_range, 267
random_structure, 170	generate_ising1_long_range_periodic, 268
hnco::function::OnBudgetFunction	random, 269
eval, 173	hnco::function::real, 36
incremental_eval, 173	hnco::function::real::DyadicRealRepresentation
update, 174	DyadicRealRepresentation, 69
hnco::function::OneMax	hnco::function::real::ParsedRealMultivariateFunction
get_maximum, 176	ParsedRealMultivariateFunction, 185
has_known_maximum, 176	hnco:: function:: real:: Real Multivariate Function Adapter
provides_incremental_evaluation, 176	RealMultivariateFunctionAdapter, 217
hnco::function::ParsedModifier	hnco::neighborhood, 37
ParsedModifier, 184	hnco::neighborhood::BernoulliProcess
hnco::function::Plateau	BernoulliProcess, 49, 50
get_maximum, 190	set_allow_stay, 50
has_known_maximum, 190	set_probability, 50
hnco::function::PriorNoise	hnco::neighborhood::HammingBall
get_maximum, 200	HammingBall, 97
has_known_maximum, 200	hnco::neighborhood::HammingSphere
provides_incremental_evaluation, 200	HammingSphere, 99
hnco::function::ProgressTracker	hnco::neighborhood::HammingSphereIterator
get_last_improvement, 203	HammingSphereIterator, 101
hnco::function::Qubo	hnco::neighborhood::MultiBitFlip
_q, 210	bernoulli_trials, 149
load, 209	MultiBitFlip, 149
hnco::function::Ridge	reservoir_sampling, 150
get_maximum, 221	hnco::neighborhood::Neighborhood
has_known_maximum, 221	map, 166
hnco::function::SixPeaks	mutate, 166
get_maximum, 229	Neighborhood, 165
has_known_maximum, 229	hnco::neighborhood::NeighborhoodIterator
hnco::function::StopOnMaximum	NeighborhoodIterator, 167
eval, 237	hnco::neighborhood::SingleBitFlipIterator
incremental_eval, 237	SingleBitFlipIterator, 226
StopOnMaximum, 236	hnco::random, 38

HncoEvaluator, 108 HypercubeIterator, 109	Model, 144 ModelParameters, 145
in a second and a second	MuCommaLambdaEa, 146
incremental_eval	hnco::algorithm::MuCommaLambdaEa, 147
hnco::function::Function, 84	MuPlusLambdaEa, 150
hnco::function::OnBudgetFunction, 173	hnco::algorithm::MuPlusLambdaEa, 152
hnco::function::StopOnMaximum, 237	MultiBitFlip, 148
hnco::function::StopOnTarget, 240	hnco::neighborhood::MultiBitFlip, 149
init_beta	multiply
hnco::algorithm::SimulatedAnnealing, 223 Injection, 110	hnco::Transvection, 250
hnco::Injection, 111	multiply_right
	hnco::Transvection, 251
is_in_range hnco, 24	mutate
is_surjective	hnco::neighborhood::Neighborhood, 166
hnco::AffineMap, 44	NearestNeighborlsingModel1, 153
hnco::LinearMap, 126	NearestNeighborIsingModel2, 156
hnco::Map, 134	Needle, 160
hnco::MapComposition, 136	Negation, 162
hnco::Permutation, 188	Neighborhood, 164
hnco::Projection, 206	hnco::neighborhood::Neighborhood, 165
hnco::Translation, 248	Neighborhood/terator, 167
hnco::TsAffineMap, 256	hnco::neighborhood::NeighborhoodIterator, 167
is_valid	NkLandscape, 168
hnco::Transvection, 249	NpsPbil, 170
IterativeAlgorithm, 112	14001 511, 170
hnco::algorithm::IterativeAlgorithm, 113	OnBudgetFunction, 172
Iterator, 115	OneMax, 174
nordior, 110	OnePlusLambdaCommaLambdaGa, 177
Jump, 116	hnco::algorithm::OnePlusLambdaComma← LambdaGa, 178
Labs, 118	OnePlusOneEa, 179
LabsMeritFactor, 119	hnco::algorithm::OnePlusOneEa, 180
LastEvaluation, 120	Ç
LeadingOnes, 120	ParameterLessPopulationPyramid, 181
LinearFunction, 122	ParsedModifier, 183
LinearMap, 125	hnco::function::ParsedModifier, 184
load	ParsedRealMultivariateFunction, 184
hnco::function::AbstractMaxSat, 41	hnco::function::real::ParsedRealMultivariate←
hnco::function::Factorization, 77	Function, 185
hnco::function::MaxNae3Sat, 138	Pbil, 186
hnco::function::Qubo, 209	perm_identity
LocalMaximum, 127	hnco, 25
LogContext, 128	perm_random
LongPath, 129	hnco, 25
Ltga, 131	Permutation, 187
	Plateau, 189
Map, 132	plus_selection
map	hnco::algorithm::Population, 197
hnco::neighborhood::Neighborhood, 166	PointValueException, 191
MapComposition, 134	Population, 192
hnco::MapComposition, 135	PriorNoise, 199
MaxNae3Sat, 137	ProgressTracker, 201
MaxSat, 138	ProgressTracker::Event, 74
maximize	ProgressTrackerContext, 203
hnco::algorithm::IterativeAlgorithm, 114	Projection, 204
MaximumReached, 136	hnco::Projection, 205
Mimic, 140	provides_incremental_evaluation
Mmas, 143	hnco::function::Cache, 60

hnco::function::Function, 85 hnco::function::FunctionController, 87	hnco::algorithm::OnePlusOneEa, 181 hnco::neighborhood::BernoulliProcess, 50 set_num_iterations
hnco::function::LinearFunction, 124 hnco::function::NearestNeighborIsingModel1, 155	hnco::algorithm::lterativeAlgorithm, 114
hnco::function::NearestNeighborIsingModel2, 159	hnco::algorithm::OnePlusOneEa, 181
hnco::function::Negation, 163 hnco::function::OneMax, 176	set_patience
hnco::function::PriorNoise, 200	hnco::algorithm::RandomLocalSearch, 212
hnco::function::WalshExpansion1, 265	set_probability hnco::neighborhood::BernoulliProcess, 50
PvAlgorithm, 206	set_reset_period
r valgoritim, 200	hnco::algorithm::hea::Hea, 106
q variation	set selection size
hnco::algorithm::hea::SpinHerding, 232	hnco::algorithm::bm_pbil::BmPbil, 58
Qubo, 208	hnco::algorithm::hea::Hea, 106
	set solution
Random, 210	hnco::algorithm::Algorithm, 47
random	SimulatedAnnealing, 222
hnco::AffineMap, 44	SingleBitFlip, 224
hnco::LinearMap, 126	SingleBitFlipIterator, 225
hnco::Transvection, 251	hnco::neighborhood::SingleBitFlipIterator, 226
hnco::TsAffineMap, 257	SinusSummationCancellation, 226
hnco::function::EqualProducts, 71	SixPeaks, 227
hnco::function::LinearFunction, 124	sparse_bit_matrix_t
hnco::function::MaxSat, 139, 140	hnco, 18
hnco::function::NearestNeighborIsingModel1, 155	sparse_bit_vector_t
hnco::function::NearestNeighborIsingModel2, 159	hnco, 18
hnco::function::NkLandscape, 170	SpinHerding, 229
hnco::function::WalshExpansion, 262	hnco::algorithm::hea::SpinHerding, 232
hnco::function::WalshExpansion1, 265	SpinMoment, 232
hnco::function::WalshExpansion2, 269	SteepestAscentHillClimbing, 234
random_non_commuting	StopOnMaximum, 235
hnco::Transvection, 251	hnco::function::StopOnMaximum, 236
random_structure	StopOnTarget, 238
hnco::function::NkLandscape, 170	hnco::function::StopOnTarget, 239
RandomLocalSearch, 211	StopWatch, 241
RandomSearch, 213	SummationCancellation, 242
RandomWalk, 214	hnco::function::SummationCancellation, 243
RealMultivariateFunction, 215	
RealMultivariateFunctionAdapter, 216	TargetReached, 244
$hnco:: function:: real:: Real Multivariate Function {\leftarrow}$	TournamentSelection, 245
Adapter, 217	Translation, 247
RealRepresentation, 218	Transvection, 248
reservoir_sampling	transvection_sequence_t
hnco::neighborhood::MultiBitFlip, 150	hnco, 18
Restart, 219	Trap, 252
Ridge, 220	hnco::function::Trap, 253
anto aval	ts_is_valid
safe_eval	hnco, 25, 26
hnco::function::Function, 85	ts_multiply
SamplingMode	hnco, 26
hnco::TsAffineMap, 256	ts_random
sbm_multiply hnco, 25	hnco, 27
	ts_random_commuting
hnco::algorithm::TournamentSelection, 246	hnco, 27 ts_random_disjoint
_	hnco, 28
set_allow_stay hnco::algorithm::GeneticAlgorithm, 96	ts_random_non_commuting
hnco::algorithm::MuCommaLambdaEa, 148	hnco, 28
hnco::algorithm::MuPlusLambdaEa, 152	ts_random_unique_destination
ooa.go	

```
hnco, 29

ts_random_unique_source
hnco, 29

TsAffineMap, 255

Umda, 257

UniformCrossover, 259
update
hnco::function::OnBudgetFunction, 174
hnco::function::StopOnMaximum, 237
hnco::function::StopOnTarget, 240
update_solution
hnco::algorithm::Algorithm, 47

WalshExpansion, 260
WalshExpansion1, 262
WalshExpansion2, 265
```