HNCO

0.12

Generated by Doxygen 1.8.13

Contents

1	Nam	nespace	Index		1
	1.1	Names	space List		1
2	Hier	archica	l Index		3
	2.1	Class I	Hierarchy		3
3	Clas	s Index			7
	3.1	Class I	List		7
4	Nam	nespace	Docume	ntation	13
	4.1	hnco N	lamespace	e Reference	13
		4.1.1	Detailed	Description	16
		4.1.2	Typedef	Documentation	16
			4.1.2.1	bit_t	17
			4.1.2.2	sparse_bit_matrix_t	17
			4.1.2.3	sparse_bit_vector_t	17
		4.1.3	Function	Documentation	17
			4.1.3.1	bm_add_rows()	17
			4.1.3.2	bm_identity()	18
			4.1.3.3	bm_invert()	18
			4.1.3.4	bm_multiply()	19
			4.1.3.5	bm_rank()	19
			4.1.3.6	bm_row_echelon_form()	19
			4.1.3.7	bm_solve()	19
			4.1.3.8	bm_solve_upper_triangular()	20

ii CONTENTS

		4.1.3.9 bv_from_vector_boo)	 21
		4.1.3.10 bv_to_vector_bool()		 21
		4.1.3.11 perm_identity()		 21
		4.1.3.12 perm_random()		 22
		4.1.3.13 sbm_multiply()		 22
	4.2	2 hnco::algorithm Namespace Reference		 22
		4.2.1 Detailed Description		 25
	4.3	3 hnco::algorithm::bm_pbil Namespace F	eference	 25
		4.3.1 Detailed Description		 25
	4.4	4 hnco::algorithm::hea Namespace Refe	ence	 25
		4.4.1 Detailed Description		 25
	4.5	5 hnco::neighborhood Namespace Refer	nce	 26
		4.5.1 Detailed Description		 26
	4.6	6 hnco::random Namespace Reference		 26
		4.6.1 Detailed Description		 26
5	Clas	ass Documentation		27
5	Clas 5.1			 27
5		1 AbstractLabs Class Reference		
5		1 AbstractLabs Class Reference		 27
5	5.1	1 AbstractLabs Class Reference		 27 28
5	5.1	1 AbstractLabs Class Reference		 27 28 28
5	5.1	AbstractLabs Class Reference		 27 28 28 29
5	5.1	1 AbstractLabs Class Reference	n	27 28 28 29 29
5	5.1	1 AbstractLabs Class Reference	n	27 28 28 29 29
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression	n	27 28 28 29 29 29
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression AdditiveGaussianNoise Class Reference	n	27 28 28 29 29 29 29
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression AdditiveGaussianNoise Class Reference 5.3.1 Detailed Description	n	27 28 29 29 29 29 29 30
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression AdditiveGaussianNoise Class Reference 5.3.1 Detailed Description 5.3.2 Member Function Documentation	n	27 28 28 29 29 29 29 29 30 31
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression AdditiveGaussianNoise Class Reference 5.3.1 Detailed Description 5.3.2 Member Function Documentat 5.3.2.1 get_maximum()	n	27 28 29 29 29 29 30 31 31
5	5.1	AbstractLabs Class Reference 5.1.1 Detailed Description AbstractMaxSat Class Reference 5.2.1 Detailed Description 5.2.2 Member Function Documentat 5.2.2.1 load() 5.2.3 Member Data Documentation 5.2.3.1 _expression AdditiveGaussianNoise Class Reference 5.3.1 Detailed Description 5.3.2 Member Function Documentat 5.3.2.1 get_maximum() 5.3.2.2 has_known_maximum	n	277 288 299 299 299 300 311 311

CONTENTS

	5.4.1	Detailed Description	33
	5.4.2	Member Function Documentation	33
		5.4.2.1 is_surjective()	33
		5.4.2.2 random()	33
5.5	Algorith	nm Class Reference	34
	5.5.1	Detailed Description	36
	5.5.2	Member Function Documentation	36
		5.5.2.1 set_solution()	36
		5.5.2.2 update_solution()	37
	5.5.3	Member Data Documentation	37
		5.5.3.1 _functions	37
5.6	Bernou	IlliProcess Class Reference	37
	5.6.1	Detailed Description	38
	5.6.2	Constructor & Destructor Documentation	38
		5.6.2.1 BernoulliProcess() [1/2]	38
		5.6.2.2 BernoulliProcess() [2/2]	39
	5.6.3	Member Function Documentation	39
		5.6.3.1 set_allow_stay()	39
		5.6.3.2 set_probability()	39
5.7	Biased	Crossover Class Reference	40
	5.7.1	Detailed Description	40
	5.7.2	Member Function Documentation	40
		5.7.2.1 breed()	40
5.8	BitHero	ling Class Reference	41
	5.8.1	Detailed Description	42
	5.8.2	Member Enumeration Documentation	42
		5.8.2.1 anonymous enum	42
5.9	BitMom	nent Struct Reference	43
	5.9.1	Detailed Description	44
5.10	BmPbil	Class Reference	44

iv CONTENTS

	5.10.1	Detailed Description	46
	5.10.2	Member Enumeration Documentation	46
		5.10.2.1 anonymous enum	46
		5.10.2.2 anonymous enum	47
		5.10.2.3 anonymous enum	47
	5.10.3	Member Function Documentation	47
		5.10.3.1 set_selection_size()	47
5.11	Cache	Class Reference	48
	5.11.1	Detailed Description	49
	5.11.2	Constructor & Destructor Documentation	49
		5.11.2.1 Cache()	49
	5.11.3	Member Function Documentation	49
		5.11.3.1 provides_incremental_evaluation()	50
5.12	CallCo	unter Class Reference	50
	5.12.1	Detailed Description	51
5.13	Compa	actGa Class Reference	51
	5.13.1	Detailed Description	53
5.14	Comple	eteSearch Class Reference	53
	5.14.1	Detailed Description	54
5.15	Crosso	ver Class Reference	54
	5.15.1	Detailed Description	54
	5.15.2	Member Function Documentation	55
		5.15.2.1 breed()	55
5.16	Decept	iveJump Class Reference	55
	5.16.1	Detailed Description	56
	5.16.2	Member Function Documentation	56
		5.16.2.1 get_maximum()	56
		5.16.2.2 has_known_maximum()	57
5.17	EqualP	Products Class Reference	57
	5.17.1	Detailed Description	58

CONTENTS

	5.17.2	Member Function Documentation	58
		5.17.2.1 random() [1/2]	58
		5.17.2.2 random() [2/2]	59
5.18	Error C	class Reference	59
	5.18.1	Detailed Description	60
5.19	Progre	ssTracker::Event Struct Reference	60
	5.19.1	Detailed Description	60
5.20	Except	ion Class Reference	61
	5.20.1	Detailed Description	61
5.21	Factoria	zation Class Reference	61
	5.21.1	Detailed Description	63
	5.21.2	Constructor & Destructor Documentation	63
		5.21.2.1 Factorization()	63
	5.21.3	Member Function Documentation	63
		5.21.3.1 load()	63
5.22	FirstAs	centHillClimbing Class Reference	64
	5.22.1	Detailed Description	65
5.23	FourPe	eaks Class Reference	65
	5.23.1	Detailed Description	66
	5.23.2	Member Function Documentation	67
		5.23.2.1 get_maximum()	67
		5.23.2.2 has_known_maximum()	67
5.24	Function	on Class Reference	68
	5.24.1	Detailed Description	69
	5.24.2	Member Function Documentation	69
		5.24.2.1 compute_walsh_transform()	69
		5.24.2.2 get_maximum()	70
		5.24.2.3 incremental_eval()	70
		5.24.2.4 provides_incremental_evaluation()	71
		5.24.2.5 safe_eval()	71

vi

5.25	Function	onController Class Reference	72
	5.25.1	Detailed Description	73
	5.25.2	Member Function Documentation	73
		5.25.2.1 provides_incremental_evaluation()	73
5.26	Function	onDecorator Class Reference	73
	5.26.1	Detailed Description	74
5.27	Function	onMapComposition Class Reference	74
	5.27.1	Detailed Description	75
	5.27.2	Constructor & Destructor Documentation	75
		5.27.2.1 FunctionMapComposition()	75
	5.27.3	Member Function Documentation	76
		5.27.3.1 get_maximum()	76
		5.27.3.2 has_known_maximum()	76
5.28	Function	onModifier Class Reference	77
	5.28.1	Detailed Description	77
5.29	Function	onPlugin Class Reference	78
	5.29.1	Detailed Description	79
	5.29.2	Constructor & Destructor Documentation	79
		5.29.2.1 FunctionPlugin()	79
5.30	Genetic	cAlgorithm Class Reference	79
	5.30.1	Detailed Description	81
	5.30.2	Constructor & Destructor Documentation	81
		5.30.2.1 GeneticAlgorithm()	81
	5.30.3	Member Function Documentation	82
		5.30.3.1 set_allow_stay()	82
5.31	Hammi	ingBall Class Reference	82
	5.31.1	Detailed Description	83
	5.31.2	Constructor & Destructor Documentation	83
		5.31.2.1 HammingBall()	83
5.32	Hammi	ingSphere Class Reference	84

CONTENTS vii

	5.32.1	Detailed Description	85
	5.32.2	Constructor & Destructor Documentation	85
		5.32.2.1 HammingSphere()	85
5.33	Hammi	ngSphereIterator Class Reference	85
	5.33.1	Detailed Description	86
	5.33.2	Constructor & Destructor Documentation	87
		5.33.2.1 HammingSphereIterator()	87
5.34	Hboa C	Class Reference	87
	5.34.1	Detailed Description	88
5.35	Hea< I	Moment, Herding > Class Template Reference	88
	5.35.1	Detailed Description	91
	5.35.2	Member Enumeration Documentation	91
		5.35.2.1 anonymous enum	91
	5.35.3	Constructor & Destructor Documentation	91
		5.35.3.1 Hea()	91
	5.35.4	Member Function Documentation	92
		5.35.4.1 set_reset_period()	92
		5.35.4.2 set_selection_size()	92
5.36	Hiff Cla	ss Reference	93
	5.36.1	Detailed Description	93
	5.36.2	Member Function Documentation	94
		5.36.2.1 get_maximum()	94
		5.36.2.2 has_known_maximum()	94
5.37	HncoE	valuator Class Reference	94
	5.37.1	Detailed Description	95
5.38	Hyperc	ubelterator Class Reference	95
	5.38.1	Detailed Description	96
5.39	Injectio	n Class Reference	96
	5.39.1	Detailed Description	97
	5.39.2	Constructor & Destructor Documentation	97

viii CONTENTS

		5.39.2.1	Injection()			 	 	 	 	 97
5.40	Iterative	eAlgorithm	Class Refer	ence		 	 	 	 	 98
	5.40.1	Detailed [Description			 	 	 	 	 99
	5.40.2	Construct	or & Destruc	tor Docume	entation .	 	 	 	 	 99
		5.40.2.1	IterativeAlgo	orithm()		 	 	 	 	 99
	5.40.3	Member F	Function Doc	umentation		 	 	 	 	 100
		5.40.3.1	maximize()			 	 	 	 	 100
		5.40.3.2	set_num_ite	erations() .		 	 	 	 	 100
5.41	Iterator	Class Ref	erence			 	 	 	 	 101
	5.41.1	Detailed [Description			 	 	 	 	 102
5.42	Jump (Class Refer	rence			 	 	 	 	 102
	5.42.1	Detailed [Description			 	 	 	 	 103
	5.42.2	Member F	Function Doc	cumentation		 	 	 	 	 103
		5.42.2.1	get_maximu	ım()		 	 	 	 	 103
		5.42.2.2	has_known	_maximum())	 	 	 	 	 103
5.43	Labs C	lass Refere	ence			 	 	 	 	 104
	5.43.1	Detailed [Description			 	 	 	 	 104
5.44	LabsM	eritFactor C	Class Refere	nce		 	 	 	 	 105
	5.44.1	Detailed [Description			 	 	 	 	 105
5.45	LastEv	aluation Cl	ass Referen	ce		 	 	 	 	 106
	5.45.1	Detailed [Description			 	 	 	 	 106
5.46	Leading	gOnes Cla	ss Reference	э		 	 	 	 	 106
	5.46.1	Detailed [Description			 	 	 	 	 107
	5.46.2	Member F	Function Doc	cumentation		 	 	 	 	 107
		5.46.2.1	get_maximu	ım()		 	 	 	 	 107
		5.46.2.2	has_known	_maximum())	 	 	 	 	 108
5.47	LinearF	Function CI	ass Referen	ce		 	 	 	 	 108
	5.47.1	Detailed [Description			 	 	 	 	 109
	5.47.2	Member F	Function Doc	cumentation		 	 	 	 	 110
		5.47.2.1	has_known	_maximum())	 	 	 	 	 110

CONTENTS

		5.47.2.2 provides_incremental_evaluation()	10
		5.47.2.3 random() [1/2]	10
		5.47.2.4 random() [2/2]	11
5.48	Linear	Map Class Reference	11
	5.48.1	Detailed Description	12
	5.48.2	Member Function Documentation	12
		5.48.2.1 is_surjective()	13
		5.48.2.2 random()	13
5.49	LocalN	aximum Class Reference	13
	5.49.1	Detailed Description	14
5.50	LogCo	ntext Class Reference	14
	5.50.1	Detailed Description	15
5.51	LongPa	ath Class Reference	15
	5.51.1	Detailed Description	16
	5.51.2	Member Function Documentation	17
		5.51.2.1 get_maximum()	17
		5.51.2.2 has_known_maximum()	17
5.52	Ltga Cl	ass Reference	18
	5.52.1	Detailed Description	18
5.53	Map C	ass Reference	19
	5.53.1	Detailed Description	19
	5.53.2	Member Function Documentation	19
		5.53.2.1 is_surjective()	19
5.54	MapCo	mposition Class Reference	20
	5.54.1	Detailed Description	20
	5.54.2	Constructor & Destructor Documentation	21
		5.54.2.1 MapComposition()	21
	5.54.3	Member Function Documentation	21
		5.54.3.1 is_surjective()	21
5.55	Maxim	umReached Class Reference	22

CONTENTS

	5.55.1 Detailed Description	122
5.56	MaxNae3Sat Class Reference	123
	5.56.1 Detailed Description	123
	5.56.2 Member Function Documentation	123
	5.56.2.1 load()	123
5.57	MaxSat Class Reference	124
	5.57.1 Detailed Description	125
	5.57.2 Member Function Documentation	125
	5.57.2.1 random() [1/2]	125
	5.57.2.2 random() [2/2]	125
5.58	Mimic Class Reference	126
	5.58.1 Detailed Description	127
5.59	Mmas Class Reference	128
	5.59.1 Detailed Description	129
5.60	Model Class Reference	129
	5.60.1 Detailed Description	130
5.61	ModelParameters Class Reference	130
	5.61.1 Detailed Description	131
5.62	MuCommaLambdaEa Class Reference	131
	5.62.1 Detailed Description	132
	5.62.2 Constructor & Destructor Documentation	132
	5.62.2.1 MuCommaLambdaEa()	132
	5.62.3 Member Function Documentation	133
	5.62.3.1 set_allow_stay()	133
5.63	MultiBitFlip Class Reference	133
	5.63.1 Detailed Description	134
	5.63.2 Constructor & Destructor Documentation	134
	5.63.2.1 MultiBitFlip()	134
	5.63.3 Member Function Documentation	134
	5.63.3.1 bernoulli_trials()	134

CONTENTS xi

		5.63.3.2 reservoir_sampling()	135
5.64	MuPlus	sLambdaEa Class Reference	135
	5.64.1	Detailed Description	136
	5.64.2	Constructor & Destructor Documentation	137
		5.64.2.1 MuPlusLambdaEa()	137
	5.64.3	Member Function Documentation	137
		5.64.3.1 set_allow_stay()	137
5.65	Neares	etNeighborlsingModel1 Class Reference	138
	5.65.1	Detailed Description	139
	5.65.2	Member Function Documentation	140
		5.65.2.1 eval()	140
		5.65.2.2 provides_incremental_evaluation()	140
		5.65.2.3 random() [1/2]	140
		5.65.2.4 random() [2/2]	141
5.66	Neares	atNeighborIsingModel2 Class Reference	141
	5.66.1	Detailed Description	143
	5.66.2	Member Function Documentation	143
		5.66.2.1 eval()	143
		5.66.2.2 provides_incremental_evaluation()	144
		5.66.2.3 random() [1/2]	144
		5.66.2.4 random() [2/2]	144
5.67	Needle	Class Reference	145
	5.67.1	Detailed Description	146
	5.67.2	Member Function Documentation	146
		5.67.2.1 get_maximum()	146
		5.67.2.2 has_known_maximum()	146
5.68	Negatio	on Class Reference	147
	5.68.1	Detailed Description	148
	5.68.2	Member Function Documentation	148
		5.68.2.1 get_maximum()	148

xii CONTENTS

		5.68.2.2 has_known_maximum()	48
		5.68.2.3 provides_incremental_evaluation()	49
5.69	Neighb	porhood Class Reference	49
	5.69.1	Detailed Description	51
	5.69.2	Constructor & Destructor Documentation	51
		5.69.2.1 Neighborhood()	51
	5.69.3	Member Function Documentation	51
		5.69.3.1 map()	51
		5.69.3.2 mutate()	52
5.70	Neighb	orhoodIterator Class Reference	52
	5.70.1	Detailed Description	53
	5.70.2	Constructor & Destructor Documentation	53
		5.70.2.1 NeighborhoodIterator()	53
5.71	NkLand	dscape Class Reference	53
	5.71.1	Detailed Description	55
	5.71.2	Member Function Documentation	55
		5.71.2.1 random() [1/2]	55
		5.71.2.2 random() [2/2]	55
		5.71.2.3 random_structure()	56
5.72	NpsPbi	il Class Reference	56
	5.72.1	Detailed Description	58
5.73	OnBud	getFunction Class Reference	58
	5.73.1	Detailed Description	59
	5.73.2	Member Function Documentation	59
		5.73.2.1 eval()	59
		5.73.2.2 incremental_eval()	60
		5.73.2.3 update()	60
5.74	OneMa	ax Class Reference	60
	5.74.1	Detailed Description	61
	5.74.2	Member Function Documentation	62

CONTENTS xiii

		5.74.2.1 get_maximum()	62
		5.74.2.2 has_known_maximum()	62
		5.74.2.3 provides_incremental_evaluation()	62
5.75	OnePlu	sLambdaCommaLambdaGa Class Reference	63
	5.75.1	Detailed Description	64
	5.75.2	Constructor & Destructor Documentation	64
		5.75.2.1 OnePlusLambdaCommaLambdaGa()	64
5.76	OnePlu	sOneEa Class Reference	65
	5.76.1	Detailed Description	66
	5.76.2	Constructor & Destructor Documentation	66
		5.76.2.1 OnePlusOneEa()	66
	5.76.3	Member Function Documentation	67
		5.76.3.1 set_allow_stay()	67
		5.76.3.2 set_num_iterations()	67
5.77	Parame	eterLessPopulationPyramid Class Reference	67
	5.77.1	Detailed Description	68
5.78	Pbil Cla	ass Reference	69
	5.78.1	Detailed Description	70
5.79	Permut	ation Class Reference	70
	5.79.1	Detailed Description	71
	5.79.2	Member Function Documentation	71
		5.79.2.1 is_surjective()	72
5.80	Plateau	Class Reference	72
	5.80.1	Detailed Description	73
	5.80.2	Member Function Documentation	73
		5.80.2.1 get_maximum()	73
		5.80.2.2 has_known_maximum()	73
5.81	PointVa	lueException Class Reference	74
	5.81.1	Detailed Description	74
5.82	Popula	ion Class Reference	75

xiv CONTENTS

	5.82.1	Detailed Description
	5.82.2	Member Function Documentation
		5.82.2.1 comma_selection() [1/2]
		5.82.2.2 comma_selection() [2/2]
		5.82.2.3 get_best_bv() [1/4]
		5.82.2.4 get_best_bv() [2/4]
		5.82.2.5 get_best_bv() [3/4]
		5.82.2.6 get_best_bv() [4/4]
		5.82.2.7 get_best_value() [1/2]
		5.82.2.8 get_best_value() [2/2]
		5.82.2.9 get_worst_bv() [1/2]
		5.82.2.10 get_worst_bv() [2/2]
		5.82.2.11 plus_selection() [1/2]
		5.82.2.12 plus_selection() [2/2]
	5.82.3	Member Data Documentation
		5.82.3.1 _compare_index_value
		5.82.3.2 _lookup
5.83	PriorNo	pise Class Reference
	5.83.1	Detailed Description
	5.83.2	Member Function Documentation
		5.83.2.1 get_maximum()
		5.83.2.2 has_known_maximum()
		5.83.2.3 provides_incremental_evaluation()
5.84	Progres	ssTracker Class Reference
	5.84.1	Detailed Description
	5.84.2	Member Function Documentation
		5.84.2.1 get_last_improvement()
5.85	Progres	ssTrackerContext Class Reference
	5.85.1	Detailed Description
5.86	Project	ion Class Reference

CONTENTS xv

	5.86.1	Detailed Description
	5.86.2	Constructor & Destructor Documentation
		5.86.2.1 Projection()
	5.86.3	Member Function Documentation
		5.86.3.1 is_surjective()
5.87	PvAlgo	orithm Class Reference
	5.87.1	Detailed Description
5.88	Qubo (Class Reference
	5.88.1	Detailed Description
	5.88.2	Member Function Documentation
		5.88.2.1 load()
	5.88.3	Member Data Documentation
		5.88.3.1 _q
5.89	Rando	m Struct Reference
	5.89.1	Detailed Description
5.90	Rando	mLocalSearch Class Reference
	5.90.1	Detailed Description
	5.90.2	Member Function Documentation
		5.90.2.1 set_patience()
5.91	Rando	mSearch Class Reference
	5.91.1	Detailed Description
5.92	Rando	mWalk Class Reference
	5.92.1	Detailed Description
5.93	Restar	t Class Reference
	5.93.1	Detailed Description
5.94	Ridge	Class Reference
	5.94.1	Detailed Description
	5.94.2	Member Function Documentation
		5.94.2.1 get_maximum()
		5.94.2.2 has_known_maximum()

xvi CONTENTS

5.95 SimulatedAnnealing Class Reference	. 202
5.95.1 Detailed Description	. 203
5.95.2 Member Function Documentation	. 203
5.95.2.1 init_beta()	. 203
5.96 SingleBitFlip Class Reference	. 204
5.96.1 Detailed Description	. 204
5.97 SingleBitFlipIterator Class Reference	. 205
5.97.1 Detailed Description	. 205
5.97.2 Constructor & Destructor Documentation	. 206
5.97.2.1 SingleBitFlipIterator()	. 206
5.98 SinusSummationCancellation Class Reference	. 206
5.98.1 Detailed Description	. 207
5.99 SixPeaks Class Reference	. 207
5.99.1 Detailed Description	. 208
5.99.2 Member Function Documentation	. 209
5.99.2.1 get_maximum()	. 209
5.99.2.2 has_known_maximum()	. 209
5.100 SpinHerding Class Reference	. 209
5.100.1 Detailed Description	. 211
5.100.2 Member Enumeration Documentation	. 211
5.100.2.1 anonymous enum	. 211
5.100.3 Constructor & Destructor Documentation	. 212
5.100.3.1 SpinHerding()	. 212
5.100.4 Member Function Documentation	. 212
5.100.4.1 q_variation()	. 212
5.101 SpinMoment Struct Reference	. 212
5.101.1 Detailed Description	. 213
5.101.2 Member Data Documentation	. 213
5.101.2.1 _second	. 214
5.102SteepestAscentHillClimbing Class Reference	. 214

CONTENTS xvii

5.102.1 Detailed Description
5.103StopOnMaximum Class Reference
5.103.1 Detailed Description
5.103.2 Constructor & Destructor Documentation
5.103.2.1 StopOnMaximum()
5.103.3 Member Function Documentation
5.103.3.1 eval()
5.103.3.2 incremental_eval()
5.103.3.3 update()
5.104StopOnTarget Class Reference
5.104.1 Detailed Description
5.104.2 Constructor & Destructor Documentation
5.104.2.1 StopOnTarget()
5.104.3 Member Function Documentation
5.104.3.1 eval()
5.104.3.2 incremental_eval()
5.104.3.3 update()
5.105StopWatch Class Reference
5.105.1 Detailed Description
5.106SummationCancellation Class Reference
5.106.1 Detailed Description
5.106.2 Constructor & Destructor Documentation
5.106.2.1 SummationCancellation()
5.106.3 Member Function Documentation
5.106.3.1 has_known_maximum()
5.107TargetReached Class Reference
5.107.1 Detailed Description
5.108TournamentSelection Class Reference
5.108.1 Detailed Description
5.108.2 Member Function Documentation

xviii CONTENTS

5.108.2.1 select()	226
5.109Translation Class Reference	227
5.109.1 Detailed Description	228
5.109.2 Member Function Documentation	228
5.109.2.1 is_surjective()	228
5.110Trap Class Reference	228
5.110.1 Detailed Description	229
5.110.2 Constructor & Destructor Documentation	229
5.110.2.1 Trap()	229
5.110.3 Member Function Documentation	230
5.110.3.1 get_maximum()	230
5.110.3.2 has_known_maximum()	230
5.111 Umda Class Reference	231
5.111.1 Detailed Description	232
5.112UniformCrossover Class Reference	232
5.112.1 Detailed Description	233
5.112.2 Member Function Documentation	233
5.112.2.1 breed()	233
5.113 Walsh Expansion Class Reference	233
5.113.1 Detailed Description	235
5.113.2 Member Function Documentation	235
5.113.2.1 random() [1/2]	235
5.113.2.2 random() [2/2]	235
5.114WalshExpansion1 Class Reference	236
5.114.1 Detailed Description	237
5.114.2 Member Function Documentation	237
5.114.2.1 has_known_maximum()	237
5.114.2.2 provides_incremental_evaluation()	238
5.114.2.3 random() [1/2]	238
5.114.2.4 random() [2/2]	238
5.115 Walsh Expansion 2 Class Reference	239
5.115.1 Detailed Description	240
5.115.2 Member Function Documentation	240
5.115.2.1 generate_ising1_long_range()	240
5.115.2.2 generate_ising1_long_range_periodic()	241
5.115.2.3 random() [1/2]	241
5.115.2.4 random() [2/2]	242
5.115.3 Member Data Documentation	242
5.115.3.1 _quadratic	242
5.116Function::WalshTransformTerm Struct Reference	242
5.116.1 Detailed Description	243
5.116.2 Member Data Documentation	243
5.116.2.1 feature	243

xix

Index 245

Chapter 1

Namespace Index

1.1 Namespace List

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

nnco	
Top-level HNCO namespace	13
hnco::algorithm	
Algorithms	22
hnco::algorithm::bm_pbil	
Boltzmann machine PBIL	25
hnco::algorithm::hea	
Herding evolutionary algorithm	25
hnco::neighborhood	
Neighborhoods for local search	26
hnco::random	
Pseudo random numbers	26

2 Namespace Index

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Algorithm
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

Hierarchical Index

ProgressTracker::Event	
Exception	61
Error	59
LastEvaluation	106
PointValueException	174
LocalMaximum	113
MaximumReached	122
TargetReached	<mark>224</mark>
Function	68
AbstractLabs	27
Labs	104
LabsMeritFactor	105
AbstractMaxSat	28
MaxNae3Sat	123
MaxSat	124
DeceptiveJump	55
EqualProducts	57
Factorization	61
FourPeaks	
FunctionDecorator	73
FunctionController	
Cache	
CallCounter	
OnBudgetFunction	158
ProgressTracker	184
StopOnMaximum	215
StopOnTarget	<mark>218</mark>
FunctionModifier	77
AdditiveGaussianNoise	30
FunctionMapComposition	74
Negation	147
PriorNoise	182
FunctionPlugin	78
Hiff	
Jump	102
LeadingOnes	
LinearFunction	108
LongPath	
NearestNeighborIsingModel1	
NearestNeighborIsingModel2	
Needle	
NkLandscape	
OneMax	
Plateau	
Qubo	
Ridge	
SixPeaks	
SummationCancellation	
SinusSummationCancellation	
Trap	
WalshExpansion	
WalshExpansion1	
WalshExpansion2	
lterator	
Hypercubelterator	
NeighborhoodIterator	
HammingSphereIterator	85

2.1 Class Hierarchy 5

SingleBitFlipIterator	205
LogContext	114
ProgressTrackerContext	186
Мар	119
AffineMap	32
Injection	96
LinearMap	
MapComposition	120
Permutation	170
Projection	
Translation	
Model	129
ModelParameters	
Neighborhood	149
MultiBitFlip	133
BernoulliProcess	37
HammingBall	82
HammingSphere	84
SingleBitFlip	204
Population	175
TournamentSelection	225
Random	193
SpinHerding	
SpinMoment	212
StopWatch	221
Function::WalshTransformTerm	242

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	27
AbstractMaxSat	
Abstract class for MaxSat-like functions	28
AdditiveGaussianNoise	
Additive Gaussian Noise	30
AffineMap	
Affine map	32
Algorithm	
Abstract search algorithm	34
BernoulliProcess	
Bernoulli process	37
BiasedCrossover	
Biased crossover	40
BitHerding	
Herding with bit features	41
BitMoment	
Moment for bit features	43
BmPbil Dr. Harris DR.	
Boltzmann machine PBIL	44
Cache	40
Cache	48
CallCounter	F.0
Call counter	50
CompactGa	E4
Compact genetic algorithm	51
CompleteSearch	53
Complete search	53
Crossover Crossover	54
DeceptiveJump	54
	55
Deceptive jump	55
Equal products	57
Equal products	5/
EIIOI	

8 Class Index

ProgressTracker::Event	
Event	60
Exception Basic exception	61
Factorization	01
Factorization	61
FirstAscentHillClimbing First ascent hill climbing	64
FourPeaks	
Four Peaks	65
Function	00
FunctionController	
Function controller	
FunctionDecorator Function decorator	73
FunctionMapComposition	
Composition of a function and a map	
FunctionModifier	
Function modifier	
FunctionPlugin	
Function plugin	78
GeneticAlgorithm Genetic algorithm	79
HammingBall	
Hamming ball	82
HammingSphere	
Hamming sphere	84
HammingSphereIterator	0.5
Hamming sphere neighborhood iterator	
Hierarchical Bayesian Optimization Algorithm	87
Hea< Moment, Herding >	
Herding evolutionary algorithm	88
Hiff	
Hierarchical if and only if	93
HncoEvaluator Evaluator for HNCO functions	0.4
Evaluator for HNCO functions	94
Hypercube iterator	95
Injection	
Injection	96
IterativeAlgorithm	
Iterative search	98
Iterator lterator over bit vectors	101
Jump	
Jump	102
Labs	
Low autocorrelation binary sequences	104
LabsMeritFactor	105
Low autocorrelation binary sequences merit factor	105
Last evaluation	106
LeadingOnes	
Leading ones	106
LinearFunction	
Linear function	108

3.1 Class List

LinearMap	
·	11
LocalMaximum Local maximum	13
LogContext	
Log context	
Long path	15
Linkage Tree Genetic Algorithm	18
Мар	19
MapComposition Map composition	20
MaximumReached	
Maximum reached	22
Max not-all-equal 3SAT	23
MAX-SAT	24
Mimic Mutual information maximizing input clustering	26
Mmas	
Model	20
Model of a Boltzmann machine	29
Parameters of a Boltzmann machine	30
MuCommaLambdaEa (mu, lambda) EA	31
MultiBitFlip	
Multi bit flip	33
(mu+lambda) EA	35
NearestNeighborIsingModel1 Nearest neighbor Ising model in one dimension	38
NearestNeighborIsingModel2 Nearest neighbor Ising model in two dimensions	41
Needle	Τ,
Needle in a haystack	45
Negation Negation	47
Neighborhood	4 0
NeighborhoodIterator	
Neighborhood iterator	52
NK landscape	53
Population-based incremental learning with negative and positive selection	56
OnBudgetFunction CallCounter with a limited number of evaluations	58
OneMax	
OnePlusLambdaCommaLambdaGa	
(1+(lambda, lambda)) genetic algorithm	63
(1+1) EA	65

10 Class Index

Paramet	erLessPopulationPyramid Parameter-less Population Pyramid	67
Pbil		
Permuta		69
Territa		70
Plateau	Plateau	72
PointValu	ueException	
		74
Population		75
PriorNois	•	75
		82
Progress		84
Progress	ProgressTracker	04
		86
Projectio		07
PvAlgori	-3	87
r vrugori		89
Qubo		
Dandan		91
Random		93
Random	LocalSearch	
		94
Random		96
Random		50
	Random walk	97
Restart	Doctors	00
Ridge	Restart	98
·go	Ridge	200
Simulate	edAnnealing	
SingleBit	· · · · · · · · · · · · · · · · · · ·	202
Singlebil	·	204
SingleBit	tFlipIterator	
0:		205
SinusSu	mmationCancellation Summation cancellation with sinus	206
SixPeak		
		207
SpinHero		209
SpinMon		.03
	Moment for spin variables	12
Steepes	tAscentHillClimbing	
StonOnN	Steepest ascent hill climbing	:14
ClopOrm	Stop on maximum	215
StopOnT	Farget	
Cton\Mo+	Stop on target	:18
StopWat	Stop watch	221
	•	

3.1 Class List

SummationCancellation
Summation cancellation
TargetReached
Target reached
TournamentSelection
Population with tournament selection
Translation
Translation
Trap
Trap
Umda
Univariate marginal distribution algorithm
UniformCrossover
Uniform crossover
WalshExpansion
Walsh expansion
WalshExpansion1
Walsh expansion of degree 1
WalshExpansion2
Walsh expansion of degree 2
Function::WalshTransformTerm
Walsh transform term

12 Class Index

Chapter 4

Namespace Documentation

4.1 hnco Namespace Reference

top-level HNCO namespace

Namespaces

· algorithm

Algorithms.

· 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.

Functions

Transpose.

```
    template<class T >

      T square (T x)
          Generic square function.

    double logistic (double x)

          Logistic function (sigmoid)
Types and functions related to bit matrices

    typedef std::vector< bit_vector_t > bit_matrix_t

          Bit matrix.

    void bm display (const bit matrix t &M, std::ostream &stream)

          Display bit matrix.

    bool bm_is_valid (const bit_matrix_t &M)

          Check whether a bit matrix is valid.

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

          Number of rows.
    • size_t bm_num_columns (const bit_matrix_t &M)
          Number of columns.

    bool bm_is_square (const bit_matrix_t &M)

          Check whether the matrix is a square matrix.

    bool bm is identity (const bit matrix t &M)

          Check whether the matrix is the identity matrix.

    bool bm_is_upper_triangular (const bit_matrix_t &M)

          Check whether the matrix is upper triangular.

    void bm_resize (bit_matrix_t &M, std::size_t num_rows, std::size_t num_columns)

          Resize a bit matrix.

    void bm_resize (bit_matrix_t &M, std::size_t num_rows)

          Resize a bit matrix and make it a square matrix.
    void bm_clear (bit_matrix_t &M)
          Clear bit matrix.

    void bm identity (bit matrix t &M)

          Set the matrix to the identity matrix.

    void bm_random (bit_matrix_t &M)

          Sample a random bit matrix.

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

          Swap two rows.

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

          Add two rows.

    void bm_row_echelon_form (bit_matrix_t &A)

          Compute a row echelon form of a matrix.

    std::size_t bm_rank (const bit_matrix_t &A)

          Compute the rank of a matrix.

    bool bm_solve (bit_matrix_t &A, bit_vector_t &b)

          Solve a linear system.

    bool bm_solve_upper_triangular (bit_matrix_t &A, bit_vector_t &b)

          Solve a linear system in upper triangular form.

    bool bm_invert (bit_matrix_t &M, bit_matrix_t &N)

          Invert a bit matrix.

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

          Multiply a bit matrix and a bit vector.

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

Types and functions related to bit

```
· typedef unsigned char bit_t
    bit_t bit_flip (bit_t b)
          Flip bit.
    • bit_t bit_random (double p)
          Sample a random bit.
Types and functions related to bit vectors
    typedef std::vector< bit_t > bit_vector_t
           Bit vector.

    typedef std::pair< bit_vector_t, double > point_value_t

           Type to represent point value pairs.

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

           Display bit vector.

    bool bv_is_valid (const bit_vector_t &x)

           Check whether the bit vector is valid.

    bool bv_is_zero (const bit_vector_t &x)

           Check whether the bit vector is zero.

    int bv_hamming_weight (const bit_vector_t &x)

          Hamming weight.

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

          Hamming weight.

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

          Hamming distance between two bit vectors.

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

           Dot product.

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

          Dot product.

    void bv_clear (bit_vector_t &x)

           Clear bit vector.

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

          Flip a single bit.

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

          Flip many bits.

    void bv_random (bit_vector_t &x)

          Sample a random bit vector.

    void bv_random (bit_vector_t &x, int k)

          Sample a random bit vector with given Hamming weight.

    void by 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 by to vector bool (const bit vector t &x, std::vector< bool > &y)

           Convert a bit vector to a bool vector.

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

           Convert a bool vector to a bit vector.

    std::size_t bv_to_size_type (const bit_vector_t &x)

           Convert a bit vector to a size t.
```

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

Convert a size_t to a bit vector.

Types and functions related to permutations

 $\bullet \ \ typedef \ std::vector < std::size_t > \underline{permutation_t}$

Permutation type.

bool perm_is_valid (const permutation_t &permutation)

Check that a vector represents a permutation.

void perm_identity (permutation_t &s)

Identity permutation.

void perm_random (permutation_t &s)

Sample a random permutation.

Types and functions related to sparse bit matrices

• typedef std::vector< sparse_bit_vector_t > sparse_bit_matrix_t

Sparse bit matrix.

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

Display sparse bit matrix.

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

Convert a bit matrix to a sparse bit matrix.

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

Multiply a sparse bit matrix and a bit vector.

Types and functions related to sparse bit vectors

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

Sparse bit vector.

void bv_flip (bit_vector_t &x, const sparse_bit_vector_t &sbv)

Flip many bits.

void sbv_display (const sparse_bit_vector_t &v, std::ostream &stream)

Display sparse bit vector.

void bv_to_sbv (const bit_vector_t &bv, sparse_bit_vector_t &sbv)

Convert a bit vector to a sparse bit vector.

4.1.1 Detailed Description

top-level HNCO namespace

Functions to be maximized.

Exceptions.

4.1.2 Typedef Documentation

4.1.2.1 bit_t

```
typedef unsigned char bit_t
```

Bit.

A single bit is represented by an unsigned 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<std::size_t> sparse_bit_vector_t
```

Sparse bit vector.

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

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

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

4.1.3 Function Documentation

4.1.3.1 bm_add_rows()

Add two rows.

Row i is added to row j.

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

4.1.3.2 bm_identity()

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

Set the matrix to the identity matrix.

Precondition

```
bm_is_square(M)
```

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

4.1.3.3 bm_invert()

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

Invert a bit matrix.

Parameters

М	input matrix
Ν	inverse matrix

Precondition

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

Returns

true if M is invertible

Warning

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

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

4.1.3.4 bm_multiply()

Multiply a bit matrix and a bit vector.

The result is y = Mx.

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

4.1.3.5 bm_rank()

Compute the rank of a matrix.

Precondition

A must be in row echelon form.

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

4.1.3.6 bm_row_echelon_form()

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

Compute a row echelon form of a matrix.

Warning

A is modified by the function.

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

4.1.3.7 bm_solve()

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

Solve a linear system.

Solve the linear equation Ax = b.

Parameters

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 170 of file bit-matrix.cc.

4.1.3.8 bm_solve_upper_triangular()

Solve a linear system in upper triangular form.

Solve the linear equation Ax = b.

Parameters

Α	Upper triangular matrix
b	Right hand side

Precondition

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

Returns

true if the system has a unique solution

Warning

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

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

4.1.3.9 bv_from_vector_bool()

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

Convert a bool vector to a bit vector.

Warning

Vectors must be of the same size.

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

4.1.3.10 bv_to_vector_bool()

Convert a bit vector to a bool vector.

Warning

Vectors must be of the same size.

Definition at line 140 of file bit-vector.cc.

4.1.3.11 perm_identity()

Identity permutation.

Warning

This function does not set the size of the permutation.

Definition at line 46 of file permutation.hh.

4.1.3.12 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.

4.1.3.13 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.

4.2 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)
```

Type and functions related to probability vectors

Check for element range.

Accumulate a bit vector.

Accumulate a bit vector.

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

Accumulate a bit vector.

void pv_average (pv_t &pv, int count)

Average.

void pv_update (pv_t &pv, double rate, const bit_vector_t &x)

Update a probability vector toward a bit vector.

void pv_update (pv_t &pv, double rate, const std::vector< double > &x)

Update a probability vector toward a probability vector.

void pv_update (pv_t &pv, double rate, const std::vector< double > &x, const std::vector< double > &y)

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

void pv_bound (pv_t &pv, double lower_bound, double upper_bound)

Bound the components of a probability vector.

4.2.1 Detailed Description

Algorithms.

4.3 hnco::algorithm::bm_pbil Namespace Reference

Boltzmann machine PBIL.

Classes

· class BmPbil

Boltzmann machine PBIL.

· class Model

Model of a Boltzmann machine.

· class ModelParameters

Parameters of a Boltzmann machine.

4.3.1 Detailed Description

Boltzmann machine PBIL.

4.4 hnco::algorithm::hea Namespace Reference

Herding evolutionary algorithm.

Classes

· class 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::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.

4.5.1 Detailed Description

Neighborhoods for local search.

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

4.6 hnco::random Namespace Reference

Pseudo random numbers.

Classes

struct Random

Random numbers.

4.6.1 Detailed Description

Pseudo random numbers.

Chapter 5

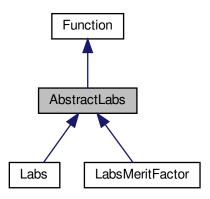
Class Documentation

5.1 AbstractLabs Class Reference

Abstract class for low autocorrelation binary sequences.

#include <hnco/functions/labs.hh>

Inheritance diagram for AbstractLabs:



Public Member Functions

• AbstractLabs (int n)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double compute_autocorrelation (const bit_vector_t &)

Compute autocorrelation.

Protected Attributes

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

5.1.1 Detailed Description

Abstract class for low autocorrelation binary sequences.

Definition at line 32 of file labs.hh.

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

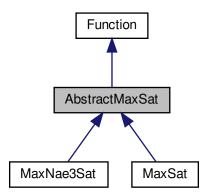
- · lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

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.

• size_t get_bv_size ()

Get bit vector size.

void display (std::ostream &stream)

Display the expression.

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

Load an instance.

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

Save an instance.

Protected Attributes

```
    std::vector< std::vector< int > > _expression
    Expression.
```

size_t _num_variables

Number of variables.

5.2.1 Detailed Description

Abstract class for MaxSat-like functions.

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

5.2.2 Member Function Documentation

5.2.2.1 load()

Load an instance.

Exceptions

Error

Reimplemented in MaxNae3Sat.

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

5.2.3 Member Data Documentation

5.2.3.1 _expression

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

Expression.

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

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

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

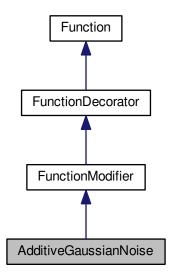
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.3 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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

Inheritance diagram for AdditiveGaussianNoise:



Public Member Functions

• AdditiveGaussianNoise (Function *function, double stddev)

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

Private Attributes

 std::normal_distribution< double > _dist Normal distribution. **Additional Inherited Members**

5.3.1 Detailed Description

Additive Gaussian Noise.

Definition at line 177 of file function-modifier.hh.

5.3.2 Member Function Documentation

5.3.2.1 get_maximum()

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

Definition at line 199 of file function-modifier.hh.

5.3.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

Definition at line 203 of file function-modifier.hh.

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

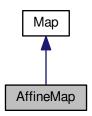
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

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)

Мар.

size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

```
• bit_matrix_t _bm
```

Bit matrix.

bit_vector_t _bv

Translation vector.

Friends

· class boost::serialization::access

5.4.1 Detailed Description

Affine map.

An affine map f from \mathbb{Z}_2^m to \mathbb{Z}_2^n is defined by f(x) = Ax + b, where A is an n x m bit matrix and b is an n-dimensional bit vector

Definition at line 257 of file map.hh.

5.4.2 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()

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

Random instance.

Parameters

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

Exceptions

Error

Definition at line 99 of file map.cc.

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

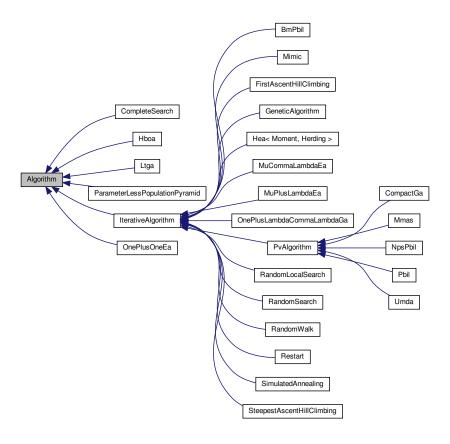
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.5 Algorithm Class Reference

Abstract search algorithm.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for Algorithm:



Public Member Functions

• Algorithm (int n)

Constructor.

virtual ∼Algorithm ()

Destructor.

Optimization

· virtual void init ()

Initialization.

• virtual void maximize ()=0

Maximize.

Getters

```
• virtual const point_value_t & get_solution ()
```

Solution.

virtual size_t get_bv_size ()

Get bit vector size.

Setters

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

Set function.

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

Set functions.

• void set_stream (std::ostream *x)

Output stream.

void set_log_context (LogContext *lc)

Set log context.

Protected Member Functions

Managing solution

```
• void random_solution ()
```

Random solution.

void set_solution (const bit_vector_t &x, double value)

Set solution.

void set_solution (const bit_vector_t &x)

Set solution.

void update_solution (const bit_vector_t &x, double value)

Update solution (strict)

void update_solution (const point_value_t &pv)

Update solution (strict)

void update_solution (const bit_vector_t &x)

Update solution (strict).

Protected Attributes

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

Parameters

Log context.

```
• 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.

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:

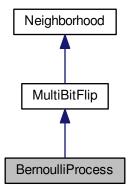
- · lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

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.

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
р	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()

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:

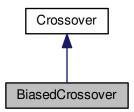
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

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	Dovvaen

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:

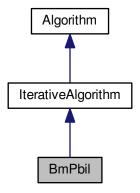
- · lib/hnco/algorithms/hea/bit-moment.hh
- lib/hnco/algorithms/hea/bit-moment.cc

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.

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:

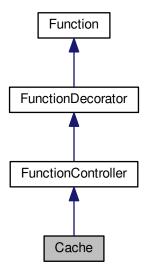
- · lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

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

::vector

bool>, for which such a function exists, before inserting it or checking its existence in the map.

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

5.11.2 Constructor & Destructor Documentation

Definition at line 382 of file function-controller.hh.

Decorated function

5.11.3 Member Function Documentation

function

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 391 of file function-controller.hh.

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

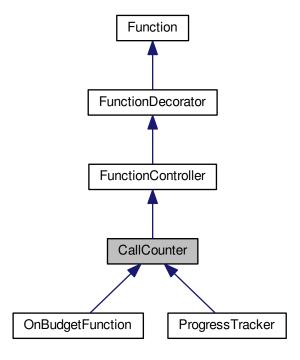
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

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 174 of file function-controller.hh.

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

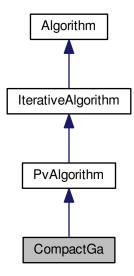
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.13 CompactGa Class Reference

Compact genetic algorithm.

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

Inheritance diagram for CompactGa:



Public Member Functions

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

Initialization.

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

std::vector < bit_vector_t > _candidates
 Candidates.

Parameters

• double <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:

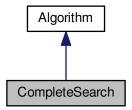
- lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

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:

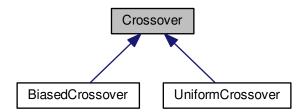
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

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:

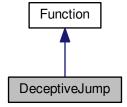
· lib/hnco/algorithms/ea/crossover.hh

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.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

```
• size_t _bv_size
```

Bit vector size.

int _gap

Gap.

5.16.1 Detailed Description

Deceptive jump.

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

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 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:

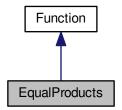
- · lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

5.17 EqualProducts Class Reference

Equal products.

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

Inheritance diagram for EqualProducts:



Public Member Functions

• EqualProducts ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Random instances

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

Random instance.

• 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

5.17.1 Detailed Description

Equal products.

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

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

Reference:

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

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

5.17.2 Member Function Documentation

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

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

Random instance.

Parameters

n	Size of bit vectors
generator	Number generator

5.18 Error Class Reference 59

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

```
5.17.2.2 random() [2/2]  \begin{tabular}{ll} $void random ( & & & \\ & & int $n$ ) [inline] \end{tabular}
```

Random instance.

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

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:

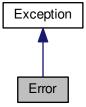
- lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

5.18 Error Class Reference

Error.

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

Inheritance diagram for Error:



Public Member Functions

```
• Error ()
```

Constructor.

• Error (const std::string &s)

Constructor.

virtual ∼Error ()

Destructor.

virtual const char * what () const

Get message.

Protected Attributes

 std::string _what Message.

5.18.1 Detailed Description

Error.

Definition at line 84 of file exception.hh.

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

· lib/hnco/exception.hh

5.19 ProgressTracker::Event Struct Reference

Event.

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

Public Attributes

• int num_evaluations

Number of evaluations.

· double value

Value.

5.19.1 Detailed Description

Event.

Definition at line 223 of file function-controller.hh.

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

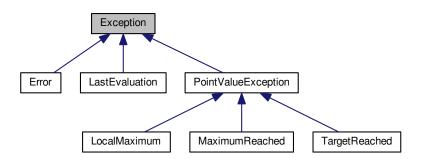
· lib/hnco/functions/decorators/function-controller.hh

5.20 Exception Class Reference

Basic exception.

#include <hnco/exception.hh>

Inheritance diagram for Exception:



5.20.1 Detailed Description

Basic exception.

Definition at line 36 of file exception.hh.

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

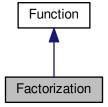
· lib/hnco/exception.hh

5.21 Factorization Class Reference

Factorization.

#include <hnco/functions/factorization.hh>

Inheritance diagram for Factorization:



Public Member Functions

· Factorization ()

Constructor.

• Factorization (const std::string number)

Constructor.

∼Factorization ()

Destructor.

void load (std::istream &stream)

Load an instance.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

· 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.

size_t _bv_size

Bit vector size.

5.21.1 Detailed Description

Factorization.

Reference:

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

```
http://gmplib.org/.
```

Definition at line 28 of file factorization.hh.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Factorization()

Constructor.

Parameters

number Number to factorize written in decimal form

Definition at line 82 of file factorization.hh.

5.21.3 Member Function Documentation

5.21.3.1 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:

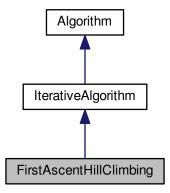
- · lib/hnco/functions/factorization.hh
- lib/hnco/functions/factorization.cc

5.22 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

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

Inheritance diagram for FirstAscentHillClimbing:



Public Member Functions

• FirstAscentHillClimbing (int n, neighborhood::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

neighborhood::Neighborhoodlterator * _neighborhood.

Neighborhood.

5.22.1 Detailed Description

First ascent hill climbing.

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

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

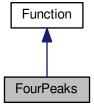
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc

5.23 FourPeaks Class Reference

Four Peaks.

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

Inheritance diagram for FourPeaks:



Public Member Functions

FourPeaks (int bv_size, int threshold)

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

double get maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

· int _threshold

Threshold.

int maximum

Maximum.

5.23.1 Detailed Description

Four Peaks.

It is defined by

```
f(x) = max\{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(111110) = 5
- f(111100) = 10 (global maximum)

Reference:

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

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

5.23.2 Member Function Documentation

5.23.2.1 get_maximum()

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

Get the global maximum.

Returns

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

Reimplemented from Function.

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

5.23.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

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

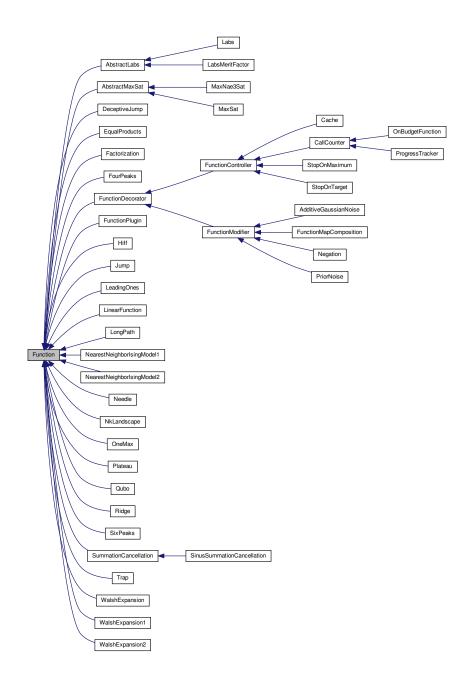
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

5.24 Function Class Reference

Function.

#include <hnco/functions/function.hh>

Inheritance diagram for Function:



Classes

• struct WalshTransformTerm

Walsh transform term.

Public Member Functions

virtual ~Function ()
 Destructor.

Information about the function

• virtual size_t get_bv_size ()=0

Get bit vector size.

virtual double get_maximum ()

Get the global maximum.

virtual bool has known maximum ()

Check for a known maximum.

virtual bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

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

Compute the Walsh transform of the function.

Evaluation

virtual double eval (const bit_vector_t &)=0

Evaluate a bit vector.

virtual double incremental_eval (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_
 t &flipped_bits)

Incremental evaluation.

virtual double safe_eval (const bit_vector_t &x)

Safely evaluate a bit vector.

virtual void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Display

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

Display

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

Describe a bit vector.

5.24.1 Detailed Description

Function.

Definition at line 41 of file function.hh.

5.24.2 Member Function Documentation

5.24.2.1 compute_walsh_transform()

```
\label{lem:compute_walsh_transform} \mbox{ ( } \\ \mbox{std::vector} < \mbox{Function::WalshTransformTerm} > \& \mbox{ terms ) } \mbox{ [virtual]}
```

Compute the Walsh transform of the function.

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

```
5.24.2.2 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

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

Definition at line 80 of file function.hh.

5.24.2.3 incremental_eval()

Incremental evaluation.

Exceptions

Error

Reimplemented in OnBudgetFunction, ProgressTracker, CallCounter, StopOnTarget, NearestNeighborIsingModel2, NearestNeighborIsingModel1, StopOnMaximum, WalshExpansion1, LinearFunction, Negation, and OneMax.

Definition at line 133 of file function.hh.

5.24.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

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

Definition at line 88 of file function.hh.

5.24.2.5 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:

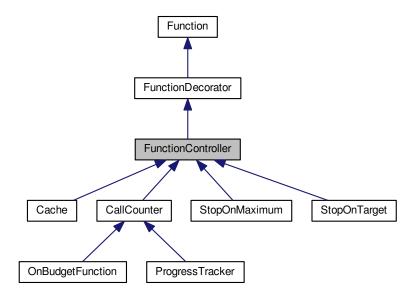
- lib/hnco/functions/function.hh
- lib/hnco/functions/function.cc

5.25 FunctionController Class Reference

Function controller.

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

Inheritance diagram for FunctionController:



Public Member Functions

• FunctionController (Function *function)

Constructor.

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

bool has_known_maximum ()

Check for a known maximum.

bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

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

Additional Inherited Members

5.25.1 Detailed Description

Function controller.

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

5.25.2 Member Function Documentation

5.25.2.1 provides_incremental_evaluation()

bool provides_incremental_evaluation () [inline], [virtual]

Check whether the function provides incremental evaluation.

Returns

true if the decorated function does

Reimplemented from Function.

Reimplemented in Cache.

Definition at line 65 of file function-controller.hh.

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

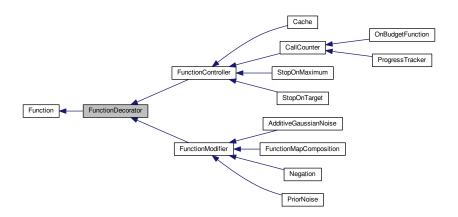
· lib/hnco/functions/decorators/function-controller.hh

5.26 FunctionDecorator Class Reference

Function decorator.

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

Inheritance diagram for FunctionDecorator:



Public Member Functions

• FunctionDecorator (Function *function)

Constructor.

Display

• void display (std::ostream &stream)

Display.

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

Describe a bit vector.

Protected Attributes

• Function * _function

Decorated function.

5.26.1 Detailed Description

Function decorator.

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

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

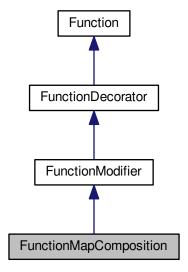
· lib/hnco/functions/decorators/function-decorator.hh

5.27 FunctionMapComposition Class Reference

Composition of a function and a map.

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

Inheritance diagram for FunctionMapComposition:



Public Member Functions

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

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

```
• size_t get_bv_size ()

Get bit vector size.
```

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

Display

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

Private Attributes

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

Image of bit vectors under the map.

Additional Inherited Members

5.27.1 Detailed Description

Composition of a function and a map.

Definition at line 107 of file function-modifier.hh.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 FunctionMapComposition()

Constructor.

Precondition

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



Error

Definition at line 122 of file function-modifier.hh.

5.27.3 Member Function Documentation

```
5.27.3.1 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

Definition at line 142 of file function-modifier.hh.

5.27.3.2 has_known_maximum()

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

Check for a known maximum.

Returns

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

Reimplemented from Function.

Definition at line 152 of file function-modifier.hh.

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

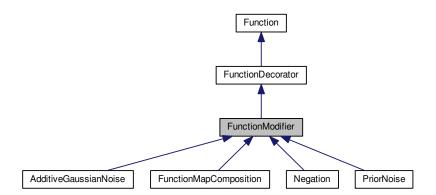
- · lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

5.28 FunctionModifier Class Reference

Function modifier.

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

Inheritance diagram for FunctionModifier:



Public Member Functions

FunctionModifier (Function *function)
 Constructor.

Additional Inherited Members

5.28.1 Detailed Description

Function modifier.

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

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

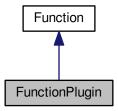
• lib/hnco/functions/decorators/function-modifier.hh

5.29 FunctionPlugin Class Reference

Function plugin.

#include <hnco/functions/plugin.hh>

Inheritance diagram for FunctionPlugin:



Public Member Functions

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

Constructor.

• ∼FunctionPlugin ()

Destructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Types

typedef double(* extern_function_t) (const bit_t *, size_t)
 Type of an extern function.

Private Attributes

• size_t _bv_size

Bit vector size.

void * _handle

Handle returned by dlopen.

extern_function_t _extern_function

Extern function.

5.29.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 FunctionPlugin()

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:

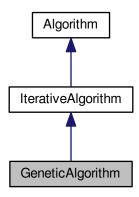
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

5.30 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



Public Member Functions

• GeneticAlgorithm (int n, int mu)

Constructor.

• void init ()

Initialization.

Setters

• void set_mutation_probability (double x)

Set the mutation probability.

• void set_crossover_probability (double x)

Set the crossover probability.

void set_tournament_size (int x)

Set the tournament size.

void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

• TournamentSelection _parents

Parents.

• TournamentSelection _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

• std::bernoulli_distribution _do_crossover

Do crossover.

• UniformCrossover _crossover

Uniform crossover.

Parameters

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

• double _crossover_probability = 0.5

Crossover probability.

• int _tournament_size = 10

Tournament size.

 bool _allow_stay = false Allow stay.

Additional Inherited Members

5.30.1 Detailed Description

Genetic algorithm.

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

Reference:

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

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

5.30.2 Constructor & Destructor Documentation

5.30.2.1 GeneticAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors
mu	Population size

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

5.30.3 Member Function Documentation

5.30.3.1 set_allow_stay()

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

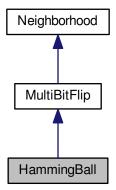
- · lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

5.31 HammingBall Class Reference

Hamming ball.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingBall:



Public Member Functions

HammingBall (int n, int r)
 Constructor.

Private Member Functions

• void sample_bits ()

Sample bits.

Private Attributes

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

Additional Inherited Members

5.31.1 Detailed Description

Hamming ball.

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

Definition at line 304 of file neighborhood.hh.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 HammingBall()

```
HammingBall (  \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.

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

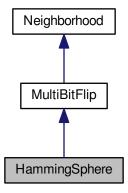
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.32 HammingSphere Class Reference

Hamming sphere.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingSphere:



Public Member Functions

• HammingSphere (int n, int r)

Constructor.

void set_radius (int r)

Set radius.

Private Member Functions

void sample_bits ()
 Sample bits.

Private Attributes

• int _radius

Radius of the sphere.

Additional Inherited Members

5.32.1 Detailed Description

Hamming sphere.

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

Definition at line 337 of file neighborhood.hh.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 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:

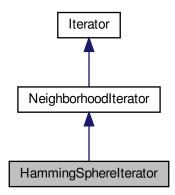
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.33 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

5.33.1 Detailed Description

Hamming sphere neighborhood iterator.

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

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.

5.34 Hboa Class Reference 87

5.33.2 Constructor & Destructor Documentation

5.33.2.1 HammingSphereIterator()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of Hamming Ball

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

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

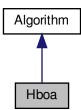
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.34 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

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

Inheritance diagram for Hboa:



Public Member Functions

• Hboa (int n)

Constructor.

• void maximize ()

Maximize.

void set_population_size (int n)

Set population size.

Private Attributes

• int _population_size = 10 Population size.

Additional Inherited Members

5.34.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 44 of file hboa.hh.

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

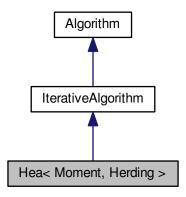
- · lib/hnco/algorithms/eda/hboa.hh
- · lib/hnco/algorithms/eda/hboa.cc

5.35 Hea < Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



Public Types

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

Public Member Functions

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

Initialization.

Setters

- void set_herding (Herding *x)
 - Set the herding algorithm.
- void set_margin (double x)
 - Set the moment margin.
- void set_selection_size (int x)
 - Set the selection size.
- void set_reset_period (int x)
 - Set the reset period.
- void set_learning_rate (double x)
 - Set the learning rate.
- void set_bound_moment (bool x)
 - Set the bound moment after update.
- void set_weight (double weight)
 - Set weight.
- void set_log_flags (const log_flags_t &lf)
 Set log flags.

Private Member Functions

```
• void iterate ()
```

Single iteration.

• void log ()

Log.

Private Attributes

Moment _target

Moment.

• Moment _selection

Moment of selected individuals.

• Moment <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.

Parameters

• double _margin

Moment margin.

• int _selection_size = 1

Selection size.

• int _reset_period = 0

Reset period.

• double _learning_rate = 1e-4

Learning rate.

• bool _bound_moment = false

Bound moment after update.

Additional Inherited Members

5.35.1 Detailed Description

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.

5.35.2 Member Enumeration Documentation

5.35.2.1 anonymous enum

anonymous enum

Enumerator

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

Definition at line 55 of file hea.hh.

5.35.3 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.

5.35.4 Member Function Documentation

5.35.4.1 set_reset_period()

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

Set the reset period.

Parameters

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

 $x \le 0$ means no reset.

Definition at line 258 of file hea.hh.

5.35.4.2 set_selection_size()

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

Set the selection size.

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

Definition at line 250 of file hea.hh.

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

· lib/hnco/algorithms/hea/hea.hh

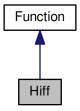
5.36 Hiff Class Reference 93

5.36 Hiff Class Reference

Hierarchical if and only if.

#include <hnco/functions/theory.hh>

Inheritance diagram for Hiff:



Public Member Functions

• Hiff (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

size_t _depth

Tree depth.

5.36.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 170 of file theory.hh.

5.36.2 Member Function Documentation

Reimplemented from Function.

Definition at line 196 of file theory.hh.

5.36.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 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

5.37 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.

5.37.1 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:

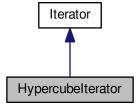
• lib/hnco/algorithms/eda/hnco-evaluator.hh

5.38 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

5.38.1 Detailed Description

Hypercube iterator.

Implemented as a simple binary adder.

Definition at line 69 of file iterator.hh.

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

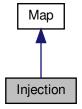
- · lib/hnco/iterator.hh
- lib/hnco/iterator.cc

5.39 Injection Class Reference

Injection.

#include <hnco/map.hh>

Inheritance diagram for Injection:



Public Member Functions

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

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

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

```
    std::vector < std::size_t > _bit_positions
    Bit positions.
```

• std::size_t _output_size Output size.

5.39.1 Detailed Description

Injection.

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

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

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

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

Definition at line 396 of file map.hh.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 Injection()

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:

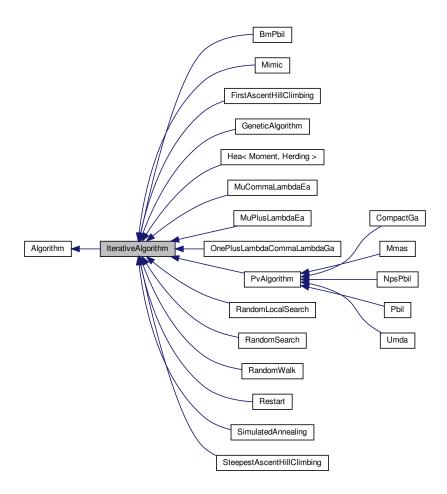
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.40 Iterative Algorithm Class Reference

Iterative search.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for IterativeAlgorithm:



Public Member Functions

```
• IterativeAlgorithm (int n)
```

Constructor.

• void maximize ()

Maximize.

Setters

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

Protected Member Functions

```
• virtual void iterate ()=0

Single iteration.
```

virtual void log ()
 Log.

Protected Attributes

· int _iteration

Current iteration.

• bool _something_to_log Something to log.

Parameters

• int _num_iterations = 0 Number of iterations.

5.40.1 Detailed Description

Iterative search.

Definition at line 169 of file algorithm.hh.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 IterativeAlgorithm()

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

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 199 of file algorithm.hh.

5.40.3 Member Function Documentation

5.40.3.1 maximize()

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

Maximize.

Inside the loop:

- call iterate()
- call log()

Warning

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

Implements Algorithm.

Definition at line 77 of file algorithm.cc.

5.40.3.2 set_num_iterations()

Set the number of iterations.

Parameters

x Number of iterations

 $x \le 0$ means indefinite

Definition at line 223 of file algorithm.hh.

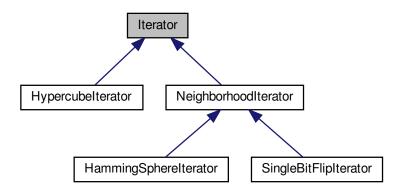
- · lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

5.41 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.

5.41.1 Detailed Description

Iterator over bit vectors.

Definition at line 34 of file iterator.hh.

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

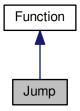
· lib/hnco/iterator.hh

5.42 Jump Class Reference

Jump.

#include <hnco/functions/jump.hh>

Inheritance diagram for Jump:



Public Member Functions

• Jump (int bv_size, int gap)

Constructor.

size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

• size_t _bv_size

Bit vector size.

int _gap

Gap.

5.42.1 Detailed Description

Jump.

Reference:

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

Definition at line 41 of file jump.hh.

5.42.2 Member Function Documentation

```
5.42.2.1 get_maximum()

double get_maximum ( ) [inline], [virtual]

Get the global maximum.
```

Returns

_bv_size

Reimplemented from Function.

Definition at line 67 of file jump.hh.

```
5.42.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 63 of file jump.hh.

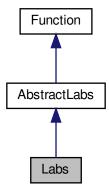
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

5.43 Labs Class Reference

Low autocorrelation binary sequences.

#include <hnco/functions/labs.hh>

Inheritance diagram for Labs:



Public Member Functions

• Labs (int n)

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.43.1 Detailed Description

Low autocorrelation binary sequences.

Reference:

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

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

Definition at line 65 of file labs.hh.

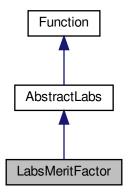
- · lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.44 LabsMeritFactor Class Reference

Low autocorrelation binary sequences merit factor.

#include <hnco/functions/labs.hh>

Inheritance diagram for LabsMeritFactor:



Public Member Functions

• LabsMeritFactor (int n)

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.44.1 Detailed Description

Low autocorrelation binary sequences merit factor.

Reference:

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

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

Definition at line 90 of file labs.hh.

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

5.45 LastEvaluation Class Reference

Last evaluation.

#include <hnco/exception.hh>

Inheritance diagram for LastEvaluation:



5.45.1 Detailed Description

Last evaluation.

Definition at line 80 of file exception.hh.

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

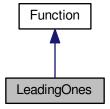
• lib/hnco/exception.hh

5.46 LeadingOnes Class Reference

Leading ones.

#include <hnco/functions/theory.hh>

Inheritance diagram for LeadingOnes:



Public Member Functions

LeadingOnes (int bv_size)

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

• size_t _bv_size

Bit vector size.

5.46.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 98 of file theory.hh.

5.46.2 Member Function Documentation

```
5.46.2.1 get_maximum()
```

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 122 of file theory.hh.

5.46.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 118 of file theory.hh.

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

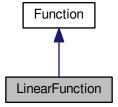
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.47 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



Public Member Functions

LinearFunction ()

Constructor.

Random instances

```
    template < class Generator >
void random (int n, Generator generator)
    Random instance.
```

• 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

```
• size_t get_bv_size ()
```

Get bit vector size.

double get_maximum ()

Get the global maximum.

bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Private Member Functions

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

Private Attributes

```
std::vector< double > _weights
Weights.
```

Friends

· class boost::serialization::access

5.47.1 Detailed Description

Linear function.

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

5.47.2 Member Function Documentation

```
5.47.2.1 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

5.47.2.2 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.

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

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

Random instance.

Parameters

n	Size of bit vectors
generator	Weight generator

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

```
5.47.2.4 random() [2/2]  \begin{tabular}{ll} $\text{void random (} & \\ & & \text{int } n \end{tabular} ) & [inline] \\ \end{tabular}
```

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:

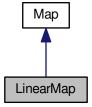
- · lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

5.48 LinearMap Class Reference

Linear map.

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

Inheritance diagram for LinearMap:



Public Member Functions

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

Random instance.

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

Мар.

size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

• bit_matrix_t _bm Bit matrix.

Friends

· class boost::serialization::access

5.48.1 Detailed Description

Linear map.

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

Definition at line 193 of file map.hh.

5.48.2 Member Function Documentation

5.48.2.1 is_surjective()

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

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 90 of file map.cc.

5.48.2.2 random()

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

Random instance.

Parameters

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:

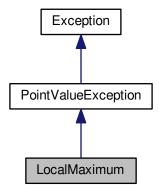
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.49 LocalMaximum Class Reference

Local maximum.

#include <hnco/exception.hh>

Inheritance diagram for LocalMaximum:



Public Member Functions

LocalMaximum (const point_value_t &pv)
 Const.

Additional Inherited Members

5.49.1 Detailed Description

Local maximum.

Definition at line 71 of file exception.hh.

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

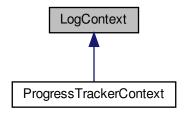
• lib/hnco/exception.hh

5.50 LogContext Class Reference

Log context.

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

Inheritance diagram for LogContext:



Public Member Functions

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

5.50.1 Detailed Description

Log context.

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

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

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

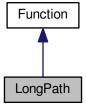
• lib/hnco/algorithms/log-context.hh

5.51 LongPath Class Reference

Long path.

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

Inheritance diagram for LongPath:



Public Member Functions

• LongPath (int bv_size, int prefix_length)

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

```
    size_t get_bv_size ()
```

Get bit vector size.

• bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

· int _prefix_length

Prefix length.

5.51.1 Detailed Description

Long path.

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

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

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

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

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

References:

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

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.51.2 Member Function Documentation

5.51.2.1 get_maximum()

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

Get the global maximum.

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

Exceptions

Error

Reimplemented from Function.

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

5.51.2.2 has_known_maximum()

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

Check for a known maximum.

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

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.

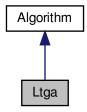
- · lib/hnco/functions/long-path.hh
- lib/hnco/functions/long-path.cc

5.52 Ltga Class Reference

Linkage Tree Genetic Algorithm.

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

Inheritance diagram for Ltga:



Public Member Functions

• Ltga (int n)

Constructor.

• void maximize ()

Maximize.

• void set_population_size (int n)

Set population size.

Private Attributes

• int _population_size = 10 Population size.

Additional Inherited Members

5.52.1 Detailed Description

Linkage Tree Genetic Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 42 of file Itga.hh.

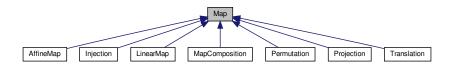
- lib/hnco/algorithms/eda/ltga.hh
- · lib/hnco/algorithms/eda/ltga.cc

5.53 Map Class Reference

Мар.

#include <hnco/map.hh>

Inheritance diagram for Map:



Public Member Functions

virtual ∼Map ()

Destructor.

virtual void map (const bit_vector_t &input, bit_vector_t &output)=0

Мар.

virtual size_t get_input_size ()=0

Get input size.

virtual size_t get_output_size ()=0

Get output size.

• virtual bool is_surjective ()

Check for surjective map.

5.53.1 Detailed Description

Мар.

Definition at line 39 of file map.hh.

5.53.2 Member Function Documentation

```
5.53.2.1 is_surjective()
```

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

Check for surjective map.

Returns

false

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

Definition at line 59 of file map.hh.

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

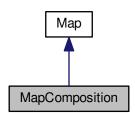
· lib/hnco/map.hh

5.54 MapComposition Class Reference

Map composition.

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

Inheritance diagram for MapComposition:



Public Member Functions

• MapComposition ()

Default constructor.

MapComposition (Map *outer, Map *inner)

Constructor.

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

Мар.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

Map * _outer

Outer map.

• Map * _inner

Inner map.

bit_vector_t _bv

Temporary bit vector.

5.54.1 Detailed Description

Map composition.

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

Definition at line 327 of file map.hh.

5.54.2 Constructor & Destructor Documentation

5.54.2.1 MapComposition()

Constructor.

Parameters

outer	outer map
inner	inner map

Precondition

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

Definition at line 351 of file map.hh.

5.54.3 Member Function Documentation

5.54.3.1 is_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 375 of file map.hh.

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

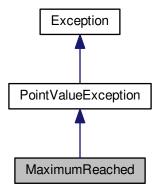
· lib/hnco/map.hh

5.55 MaximumReached Class Reference

Maximum reached.

#include <hnco/exception.hh>

Inheritance diagram for MaximumReached:



Public Member Functions

MaximumReached (const point_value_t &pv)
 Constructor.

Additional Inherited Members

5.55.1 Detailed Description

Maximum reached.

Definition at line 53 of file exception.hh.

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

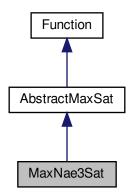
• lib/hnco/exception.hh

5.56 MaxNae3Sat Class Reference

Max not-all-equal 3SAT.

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

Inheritance diagram for MaxNae3Sat:



Public Member Functions

MaxNae3Sat ()

Default constructor.

void load (std::istream &stream)

Load an instance.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.56.1 Detailed Description

Max not-all-equal 3SAT.

Reference:

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

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

5.56.2 Member Function Documentation

```
5.56.2.1 load() \label{eq:condition} \mbox{void load (} \\ \mbox{std::istream & $stream$ ) [virtual]}
```

Load an instance.

Exceptions

Error

Reimplemented from AbstractMaxSat.

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

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

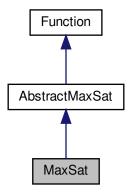
- · lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.57 MaxSat Class Reference

MAX-SAT.

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

Inheritance diagram for MaxSat:



Public Member Functions

• MaxSat ()

Default constructor.

• void random (int n, int k, int c)

Random instance.

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

Random instance with satisfiable expression.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

Additional Inherited Members

5.57.1 Detailed Description

MAX-SAT.

Reference:

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

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

5.57.2 Member Function Documentation

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:

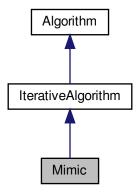
- · lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.58 Mimic Class Reference

Mutual information maximizing input clustering.

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

Inheritance diagram for Mimic:



Public Member Functions

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

Initialization.

Setters

void set_selection_size (int x)
 Set the selection size.

5.58 Mimic Class Reference 127

Protected Member Functions

• void iterate ()

Single iteration.

void sample (bit_vector_t &bv)

Sample a bit vector.

void compute_conditional_entropy (std::size_t index)

Compute conditional entropy.

void update_model ()

Update model.

Protected Attributes

Population _population

Population.

• permutation_t _permutation

Permutation.

std::array< pv_t, 2 > _parameters

Model parameters.

pv_t _mean

Mean of selected bit vectors.

std::vector< double > _entropies

Conditional entropies.

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

Contingency table.

· double _lower_bound

Lower bound of probability.

· double _upper_bound

Upper bound of probability.

Parameters

• int _selection_size Selection size.

5.58.1 Detailed Description

Mutual information maximizing input clustering.

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

Reference:

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

Definition at line 54 of file mimic.hh.

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

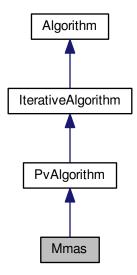
- · lib/hnco/algorithms/eda/mimic.hh
- lib/hnco/algorithms/eda/mimic.cc

5.59 Mmas Class Reference

Max-min ant system.

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

Inheritance diagram for Mmas:



Public Member Functions

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

Initialization.

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

5.60 Model Class Reference 129

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.
```

5.59.1 Detailed Description

Max-min ant system.

Reference:

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

Definition at line 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

5.60 Model Class Reference

Model of a Boltzmann machine.

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

Public Member Functions

```
• Model (int n)
```

Constructor.

• void init ()

Initialize.

· void reset_mc ()

Reset Markov chain.

• void gibbs sampler (size ti)

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.

5.60.1 Detailed Description

Model of a Boltzmann machine.

Definition at line 75 of file model.hh.

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

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

5.61 ModelParameters Class Reference

Parameters of a Boltzmann machine.

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

Public Member Functions

• ModelParameters (int n)

Constructor.

void init ()

Initialize.

void add (const bit_vector_t &x)

Add a bit_vector_t.

· void average (int count)

Compute averages.

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

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

• double norm infinite ()

Infinite norm of the parameters.

• double norm_I1 ()

I1 norm of the parameters

Private Attributes

```
\bullet \quad \mathsf{std} :: \mathsf{vector} < \mathsf{std} :: \mathsf{vector} < \mathsf{double} >> \_\mathsf{weight}
```

Weights.

std::vector< double > _bias

Bias.

Friends

· class Model

5.61.1 Detailed Description

Parameters of a Boltzmann machine.

Definition at line 36 of file model.hh.

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

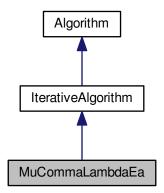
- · lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

5.62 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



Public Member Functions

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

Constructor.

· void init ()

Initialization.

Setters

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

Set the flag _allow_stay.

Private Member Functions

• void iterate ()

Single iteration.

Private Attributes

· Population _parents

Parents.

• Population _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

std::uniform_int_distribution< int > _select_parent

Select parent.

Parameters

• double _mutation_probability

Mutation probability.

 bool <u>allow_stay</u> = false Allow stay.

Allow Stay.

Additional Inherited Members

5.62.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.62.2 Constructor & Destructor Documentation

5.62.2.1 MuCommaLambdaEa()

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

Constructor.

Parameters

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

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

5.62.3 Member Function Documentation

5.62.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

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

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

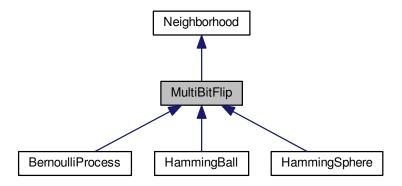
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- · lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

5.63 MultiBitFlip Class Reference

Multi bit flip.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for MultiBitFlip:



Public Member Functions

• MultiBitFlip (int n)

Constructor.

Protected Member Functions

void bernoulli_trials (int k)

Sample a given number of bits using Bernoulli trials.

void reservoir_sampling (int k)

Sample a given number of bits using resevoir sampling.

Additional Inherited Members

5.63.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 MultiBitFlip()

```
\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.

5.63.3 Member Function Documentation

5.63.3.1 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.

5.63.3.2 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:

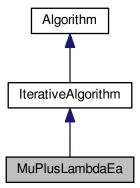
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.64 MuPlusLambdaEa Class Reference

(mu+lambda) EA.

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

Inheritance diagram for MuPlusLambdaEa:



Public Member Functions

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

Constructor.

· void init ()

Initialization.

Setters

```
• void set_mutation_probability (double x)
```

```
Set the mutation probability.
```

void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

· Population _parents

Parents.

• Population _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

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

Select parent.

Parameters

```
· double mutation probability
```

Mutation probability.

• bool <u>_allow_stay</u> = false

Allow stay.

Additional Inherited Members

5.64.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.64.2 Constructor & Destructor Documentation

5.64.2.1 MuPlusLambdaEa()

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.

5.64.3 Member Function Documentation

5.64.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

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

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

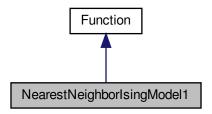
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

5.65 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.

Random instances

template < class CouplingGen , class FieldGen > void random (int n, CouplingGen coupling_gen, FieldGen field_gen)

Random instance.

• 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

• size_t 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 serialize (Archive & ar, const unsigned int version)

Serialize.

· 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

5.65.1 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.

5.65.2 Member Function Documentation

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.65.2.2 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 163 of file nearest-neighbor-ising-model-1.hh.

Random instance.

Parameters

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

Definition at line 113 of file nearest-neighbor-ising-model-1.hh.

```
5.65.2.4 random() [2/2]  \begin{tabular}{ll} void random ( & int $n$ ) [inline] \end{tabular}
```

Random instance.

The weights are sampled from the normal distribution.

Parameters

```
n Size of bit vector
```

Definition at line 129 of file nearest-neighbor-ising-model-1.hh.

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

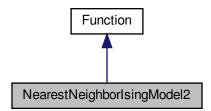
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.cc

5.66 NearestNeighborlsingModel2 Class Reference

Nearest neighbor Ising model in two dimensions.

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

Inheritance diagram for NearestNeighborlsingModel2:



Public Member Functions

· NearestNeighborIsingModel2 ()

Constructor.

void set periodic boundary conditions (bool x)

Set periodic boundary conditions.

void display (std::ostream &stream)

Display.

Random instances

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

· 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

• size t 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 serialize (Archive & ar, const unsigned int version)

Serialize.

• 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

5.66.1 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.

5.66.2 Member Function Documentation

```
5.66.2.1 eval()
```

```
double eval ( {\tt const\ bit\_vector\_t\ \&\ s\ )} \quad [{\tt virtual}]
```

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.66.2.2 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 180 of file nearest-neighbor-ising-model-2.hh.

5.66.2.3 random() [1/2]

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

Random instance.

Parameters

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

Definition at line 120 of file nearest-neighbor-ising-model-2.hh.

```
5.66.2.4 random() [2/2]
```

```
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 140 of file nearest-neighbor-ising-model-2.hh.

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

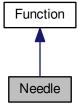
- lib/hnco/functions/ising/nearest-neighbor-ising-model-2.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-2.cc

5.67 Needle Class Reference

Needle in a haystack.

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

Inheritance diagram for Needle:



Public Member Functions

- Needle (int bv_size)
 - Constructor.
- size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

• size_t _bv_size

Bit vector size.

5.67.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 134 of file theory.hh.

5.67.2 Member Function Documentation

```
5.67.2.1 get_maximum()
```

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

Get the global maximum.

Returns

1

Reimplemented from Function.

Definition at line 158 of file theory.hh.

```
5.67.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 154 of file theory.hh.

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

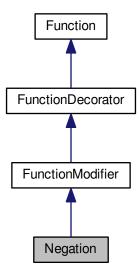
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.68 Negation Class Reference

Negation.

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

Inheritance diagram for Negation:



Public Member Functions

• Negation (Function *function)

Constructor.

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

Additional Inherited Members

5.68.1 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.

5.68.2 Member Function Documentation

```
5.68.2.1 get_maximum()

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

Definition at line 77 of file function-modifier.hh.

```
5.68.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

Definition at line 81 of file function-modifier.hh.

5.68.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

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

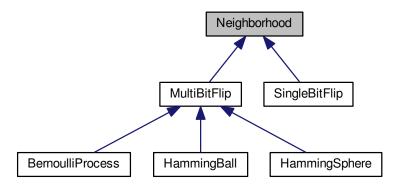
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

5.69 Neighborhood Class Reference

Neighborhood.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for Neighborhood:



Public Member Functions

Neighborhood (int n)

Constructor.

• virtual ∼Neighborhood ()

Destructor.

virtual void set_origin (const bit_vector_t &x)

Set the origin.

virtual const bit_vector_t & get_origin ()

Get the origin.

virtual const bit_vector_t & get_candidate ()

Get the candidate bit vector.

virtual const sparse_bit_vector_t & get_flipped_bits ()

Get flipped bits.

• virtual void propose ()

Propose a candidate bit vector.

· virtual void keep ()

Keep the candidate bit vector.

• virtual void forget ()

Forget the candidate bit vector.

• virtual void mutate (bit_vector_t &bv)

Mutate.

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

Мар.

Protected Member Functions

• virtual void sample_bits ()=0

Sample bits.

Protected Attributes

· bit_vector_t _origin

Origin of the neighborhood.

• bit_vector_t _candidate

candidate bit vector

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

Uniform index distribution.

• sparse_bit_vector_t _flipped_bits

Flipped bits.

5.69.1 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.

5.69.2 Constructor & Destructor Documentation

5.69.2.1 Neighborhood()

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

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 86 of file neighborhood.hh.

5.69.3 Member Function Documentation

Мар.

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

Parameters

input	Input bit vector
output	Output bit vector

Definition at line 148 of file neighborhood.hh.

5.69.3.2 mutate()

Mutate.

In-place mutation of the bit vector.

Parameters

Definition at line 134 of file neighborhood.hh.

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

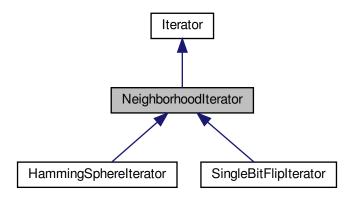
• lib/hnco/neighborhoods/neighborhood.hh

5.70 NeighborhoodIterator Class Reference

Neighborhood iterator.

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

 $Inheritance\ diagram\ for\ Neighborhood Iterator:$



Public Member Functions

NeighborhoodIterator (int n)

Constructor.

virtual void set_origin (const bit_vector_t &x)
 Set origin.

Additional Inherited Members

5.70.1 Detailed Description

Neighborhood iterator.

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

5.70.2 Constructor & Destructor Documentation

5.70.2.1 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:

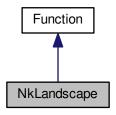
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.71 NkLandscape Class Reference

NK landscape.

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

Inheritance diagram for NkLandscape:



Public Member Functions

• NkLandscape ()

Default constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

· void display (std::ostream &stream)

Display.

Random instances

 $\begin{tabular}{ll} \bullet & template < class Generator > \\ & void \begin{tabular}{ll} void \begin{tabular}{ll} random \end{tabular} (int n, int k, Generator generator) \end{tabular}$

Random instance.

• 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

5.71.1 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.

5.71.2 Member Function Documentation

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

Random instance.

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.

5.71.2.2 random() [2/2]

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

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.

5.71.2.3 random_structure()

```
void random_structure (  \qquad \qquad \text{int } n, \\ \qquad \qquad \text{int } k \;) \quad [\text{private}]
```

Random structue.

Parameters

n	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:

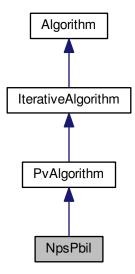
- · lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

5.72 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.

5.72.1 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:

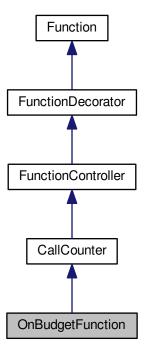
- lib/hnco/algorithms/pv/nps-pbil.hh
- · lib/hnco/algorithms/pv/nps-pbil.cc

5.73 OnBudgetFunction Class Reference

CallCounter with a limited number of evaluations.

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

 $Inheritance\ diagram\ for\ On Budget Function:$



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

5.73.1 Detailed Description

CallCounter with a limited number of evaluations.

Definition at line 318 of file function-controller.hh.

5.73.2 Member Function Documentation

```
5.73.2.1 eval()
```

Evaluate a bit vector.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.73.2.2 incremental_eval()

Incremental evaluation.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.73.2.3 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:

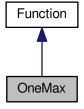
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.74 OneMax Class Reference

OneMax.

#include <hnco/functions/theory.hh>

Inheritance diagram for OneMax:



Public Member Functions

OneMax (int bv_size)
 Constructor.

Information about the function

- size_t get_bv_size ()
 - Get bit vector size.
- double get_maximum ()

Get the global maximum.

- bool has_known_maximum ()
 - Check for a known maximum.
- bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

- double eval (const bit_vector_t &)
 - Evaluate a bit vector.
- double incremental_eval (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits)

 Incremental evaluation.

Private Attributes

size_t _bv_size
 Bit vector size.

5.74.1 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.

5.74.2 Member Function Documentation

```
5.74.2.1 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.
5.74.2.2 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.
5.74.2.3 provides_incremental_evaluation()
bool provides_incremental_evaluation ( ) [inline], [virtual]
Check whether the function provides incremental evaluation.
Returns
     true
Reimplemented from Function.
```

• lib/hnco/functions/theory.cc

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

· lib/hnco/functions/theory.hh

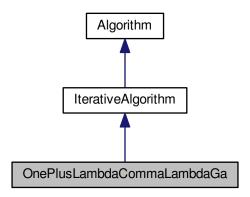
Definition at line 71 of file theory.hh.

5.75 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

5.75.1 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.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 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:

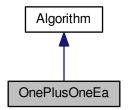
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

5.76 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

5.76.1 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.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 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.

5.76.3 Member Function Documentation

5.76.3.1 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 127 of file one-plus-one-ea.hh.

5.76.3.2 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:

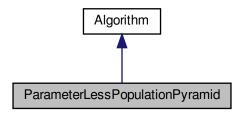
• lib/hnco/algorithms/ea/one-plus-one-ea.hh

5.77 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

5.77.1 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

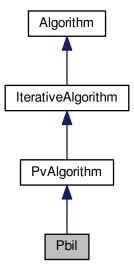
5.78 Pbil Class Reference 169

5.78 Pbil Class Reference

Population-based incremental learning.

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

Inheritance diagram for Pbil:



Public Member Functions

- Pbil (int n, int population_size)

 Constructor.
- void init ()

Initialization.

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

• Population _population

Population.

pv_t _mean

Mean of selected bit vectors.

Parameters

```
• int _selection_size = 1
```

Selection size.

• double _learning_rate = 1e-3

Learning rate.

5.78.1 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:

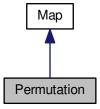
- · lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

5.79 Permutation Class Reference

Permutation.

#include <hnco/map.hh>

Inheritance diagram for Permutation:



Public Member Functions

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

Random instance.

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

Мар.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

Friends

· class boost::serialization::access

5.79.1 Detailed Description

Permutation.

A permutation is a linear map f from Z_2^n to itself defined by f(x)=y, where $y_i=x_{\sigma_i}$ and σ is a permutation of 0, 1, ..., n - 1.

Definition at line 132 of file map.hh.

5.79.2 Member Function Documentation

5.79.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 183 of file map.hh.

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

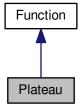
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.80 Plateau Class Reference

Plateau.

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

Inheritance diagram for Plateau:



Public Member Functions

• Plateau (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

```
    size_t _bv_size
    Bit vector size.
```

5.80.1 Detailed Description

Plateau.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 244 of file theory.hh.

5.80.2 Member Function Documentation

```
5.80.2.1 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.

5.80.2.2 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:

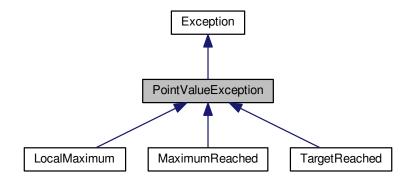
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.81 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.

5.81.1 Detailed Description

Point-value exception.

Definition at line 39 of file exception.hh.

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

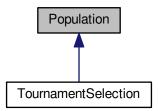
• lib/hnco/exception.hh

5.82 Population Class Reference

Population.

#include <hnco/algorithms/population.hh>

Inheritance diagram for Population:



Public Types

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

Public Member Functions

• Population (int population_size, int n)

Constructor.

 std::size_t size () const Size.

• void random ()

Initialize the population with random bit vectors.

Get bit vectors for non const populations

• bit_vector_t & get_bv (int i)

Get a bit vector.

• bit_vector_t & get_best_bv ()

Get best bit vector.

bit_vector_t & get_best_bv (int i)

Get best bit vector.

• bit_vector_t & get_worst_bv (int i)

Get worst bit vector.

Get bit vectors for const populations

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

Get best bit vector.

• const bit_vector_t & get_best_bv (int i) const

Get best bit vector.

const bit_vector_t & get_worst_bv (int i) const

Get worst bit vector.

Get sorted values

• double get_best_value (int i) const

Get best value.

double get_best_value () const

Get best value.

Evaluation and sorting

void eval (function::Function *function)

Evaluate the population.

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

Parallel evaluation of the population.

· void sort ()

Sort the lookup table.

void partial_sort (int selection_size)

Partially sort the lookup table.

• void shuffle ()

Shuffle the lookup table.

Selection

· void plus selection (const Population &offsprings)

Plus selection.

• void plus selection (Population &offsprings)

Plus selection.

• void comma_selection (const Population &offsprings)

Comma selection.

• void comma_selection (Population &offsprings)

Comma selection.

Protected Attributes

std::vector< bit_vector_t > _bvs

Bit vectors.

std::vector< index_value_t > _lookup

Lookup table.

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

Binary operator for comparing index-value pairs.

5.82.1 Detailed Description

Population.

Definition at line 36 of file population.hh.

5.82.2 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.

```
5.82.2.2 comma_selection() [2/2]
void comma_selection (
```

Comma selection.

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

Precondition

Offspring population must be partially sorted.

Population & offsprings)

Warning

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

Definition at line 126 of file population.cc.

```
5.82.2.3 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.

```
5.82.2.5 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.

```
5.82.2.6 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.

```
5.82.2.7 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.

```
5.82.2.8 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.

```
5.82.2.9 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.

```
5.82.2.10 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.

5.82.3 Member Data Documentation

5.82.3.1 _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.

5.82.3.2 _lookup

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

Lookup table.

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

Definition at line 54 of file population.hh.

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

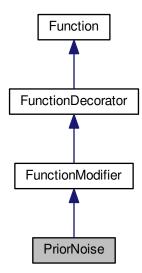
- · lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

5.83 PriorNoise Class Reference

Prior noise.

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

Inheritance diagram for PriorNoise:



Public Member Functions

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

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

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

Private Attributes

neighborhood::Neighborhood * _neighborhood
 Neighborhood.

bit_vector_t _noisy_bv

Noisy bit vector.

Additional Inherited Members

5.83.1 Detailed Description

Prior noise.

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

5.83.2 Member Function Documentation

5.83.2.1 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.

5.83.2.2 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.

5.83.2.3 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:

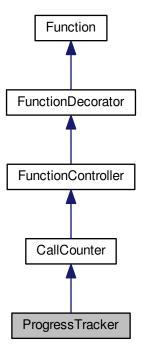
- · lib/hnco/functions/decorators/prior-noise.hh
- lib/hnco/functions/decorators/prior-noise.cc

5.84 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.

5.84.1 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 217 of file function-controller.hh.

5.84.2 Member Function Documentation

5.84.2.1 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 291 of file function-controller.hh.

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

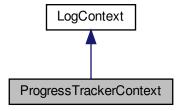
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.85 ProgressTrackerContext Class Reference

Log context for ProgressTracker.

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

Inheritance diagram for ProgressTrackerContext:



Public Member Functions

 $\bullet \quad Progress Tracker Context \ (hnco::function::Progress Tracker * pt)\\$

Constructor.

• std::string get_context ()

Get context.

Private Attributes

hnco::function::ProgressTracker * _pt

Progress tracker.

5.85.1 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:

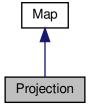
· lib/hnco/algorithms/log-context.hh

5.86 Projection Class Reference

Projection.

#include <hnco/map.hh>

Inheritance diagram for Projection:



Public Member Functions

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

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

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

```
    std::vector < std::size_t > _bit_positions
    Bit positions.
```

• std::size_t _input_size Input size.

5.86.1 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 \mathbb{Z}_2^n to \mathbb{Z}_2^m , where $n \geq m$, is defined by f(x) = y, where, for all $j \in \{1, 2, \dots, m\}$, $y_j = x_{i_j}$.

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

Definition at line 452 of file map.hh.

5.86.2 Constructor & Destructor Documentation

5.86.2.1 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.

5.86.3 Member Function Documentation

5.86.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 490 of file map.hh.

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

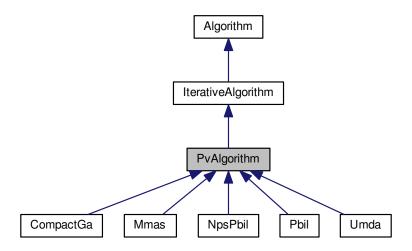
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.87 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 _upper_bound

5.88 Qubo Class Reference 191

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.

5.87.1 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:

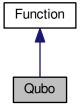
- lib/hnco/algorithms/pv/pv-algorithm.hh
- · lib/hnco/algorithms/pv/pv-algorithm.cc

5.88 Qubo Class Reference

Quadratic unconstrained binary optimization.

#include <hnco/functions/qubo.hh>

Inheritance diagram for Qubo:



Public Member Functions

• Qubo ()

Constructor.

void load (std::istream &stream)

Load an instance.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Attributes

std::vector< std::vector< double >> _q
 Matrix.

5.88.1 Detailed Description

Quadratic unconstrained binary optimization.

Its expression is of the form $f(x) = \sum_i Q_{ii} x_i + \sum_{i < j} Q_{ij} x_i x_j = x^T Q x$, where Q is an n x n upper-triangular matrix.

Qubo is the problem addressed by qbsolv. Here is its description as given on github:

Qbsolv, a decomposing solver, finds a minimum value of a large quadratic unconstrained binary optimization (Q← UBO) problem by splitting it into pieces solved either via a D-Wave system or a classical tabu solver.

There are some differences between WalshExpansion2 and Qubo:

- WalshExpansion2 maps 0/1 variables into -1/1 variables whereas Qubo directly deals with binary variables.
- Hence, there is a separate linear part in WalshExpansion2 whereas the linear part in Qubo stems from the diagonal elements of the given matrix.

qbsolv aims at minimizing quadratic functions whereas hnco algorithms aim at maximizing them. Hence Qubo::load negates all elements so that maximizing the resulting function is equivalent to minimizing the original Qubo.

References:

Michael Booth, Steven P. Reinhardt, and Aidan Roy. 2017. Partitioning Optimization Problems for Hybrid Classical/Quantum Execution. Technical Report. D-Wave.

```
https://github.com/dwavesystems/qbsolv
```

```
http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html
```

Definition at line 74 of file qubo.hh.

5.88.2 Member Function Documentation

Load an instance.

Exceptions

Error

Definition at line 35 of file qubo.cc.

5.88.3 Member Data Documentation

```
5.88.3.1 _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

5.89 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.

5.89.1 Detailed Description

Random numbers.

Definition at line 33 of file random.hh.

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

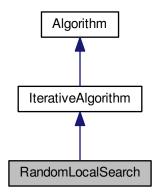
- · lib/hnco/random.hh
- · lib/hnco/random.cc

5.90 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.

5.90.1 Detailed Description

Random local search.

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

5.90.2 Member Function Documentation

5.90.2.1 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:

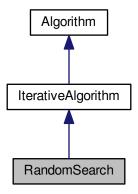
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

5.91 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

5.91.1 Detailed Description

Random search.

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

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

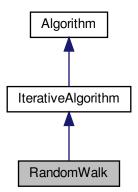
- lib/hnco/algorithms/random-search.hh
- · lib/hnco/algorithms/random-search.cc

5.92 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

5.92.1 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:

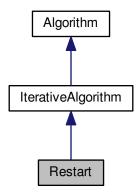
- · lib/hnco/algorithms/ls/random-walk.hh
- lib/hnco/algorithms/ls/random-walk.cc

5.93 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

5.93.1 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:

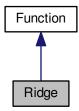
- lib/hnco/algorithms/decorators/restart.hh
- lib/hnco/algorithms/decorators/restart.cc

5.94 Ridge Class Reference

Ridge.

#include <hnco/functions/theory.hh>

Inheritance diagram for Ridge:



Public Member Functions

• Ridge (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

• size_t _bv_size

Bit vector size.

5.94.1 Detailed Description

Ridge.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 208 of file theory.hh.

5.94.2 Member Function Documentation

5.94.2.1 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.

5.94.2.2 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:

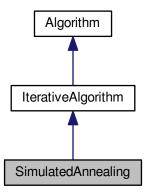
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.95 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

5.95.1 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.

5.95.2 Member Function Documentation

```
5.95.2.1 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:

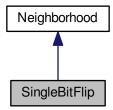
- lib/hnco/algorithms/ls/simulated-annealing.hh
- lib/hnco/algorithms/ls/simulated-annealing.cc

5.96 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

5.96.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

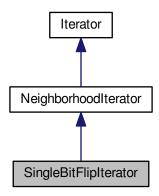
• lib/hnco/neighborhoods/neighborhood.hh

5.97 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

5.97.1 Detailed Description

Single bit flip neighborhood iterator.

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

5.97.2 Constructor & Destructor Documentation

5.97.2.1 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:

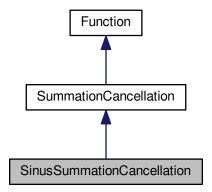
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.98 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

5.98.1 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:

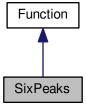
- · lib/hnco/functions/cancellation.hh
- · lib/hnco/functions/cancellation.cc

5.99 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



Public Member Functions

· SixPeaks (int bv_size, int threshold)

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

int _threshold

Threshold.

int maximum

Maximum.

5.99.1 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.

5.99.2 Member Function Documentation

```
5.99.2.1 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.

5.99.2.2 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

5.100 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, size_t i)

Derivative of q.

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

Variation of q.

void sample_rls (bit_vector_t &x)

Sample by means of random local search.

void sample_dls (bit_vector_t &x)

Sample by means of deterministic local search.

Protected Attributes

• SpinMoment _delta

Delta moment.

SpinMoment _count

Counter moment.

• permutation_t _permutation

Permutation.

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

Choose bit.

• int _time

Time.

Parameters

bool _randomize_bit_order = false

Randomize bit order.

• int _sampling_method = SAMPLE_GREEDY

Sampling method.

• int _num_seq_updates

Number of sequential updates per sample.

• double _weight = 1

Weight of second order moments.

5.100.1 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.

5.100.2 Member Enumeration Documentation

5.100.2.1 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.

5.100.3 Constructor & Destructor Documentation

5.100.3.1 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.

5.100.4 Member Function Documentation

5.100.4.1 q_variation()

Variation of q.

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

Definition at line 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

5.101 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.

5.101.1 Detailed Description

Moment for spin variables.

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

5.101.2 Member Data Documentation

5.101.2.1 _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:

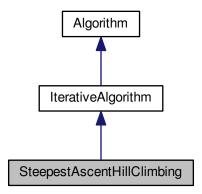
- lib/hnco/algorithms/hea/spin-moment.hh
- lib/hnco/algorithms/hea/spin-moment.cc

5.102 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.

5.102.1 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:

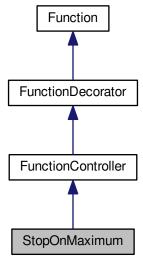
- · lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

5.103 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

5.103.1 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 91 of file function-controller.hh.

5.103.2 Constructor & Destructor Documentation

5.103.2.1 StopOnMaximum()

Constructor.

Parameters

Precondition

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

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

5.103.3 Member Function Documentation

Evaluate a bit vector.

Exceptions

MaximumReached

Implements Function.

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

5.103.3.2 incremental_eval()

Incremental evaluation.

Exceptions

```
MaximumReached
```

Reimplemented from Function.

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

5.103.3.3 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:

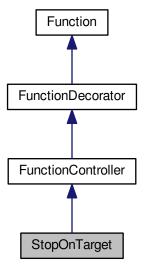
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.104 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

5.104.1 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 135 of file function-controller.hh.

5.104.2 Constructor & Destructor Documentation

5.104.2.1 StopOnTarget()

Constructor.

Parameters

function	Decorated function
target	Target

Definition at line 148 of file function-controller.hh.

5.104.3 Member Function Documentation

```
5.104.3.1 eval()
double eval (
```

const bit_vector_t & x) [virtual]

Evaluate a bit vector.

Exceptions

TargetReached

Implements Function.

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

5.104.3.2 incremental_eval()

Incremental evaluation.

Exceptions

TargetReached

Reimplemented from Function.

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

5.104.3.3 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

5.105 StopWatch Class Reference

Stop watch.

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

Public Member Functions

```
• void start ()
```

Start.

• void stop ()

Stop.

• double get_total ()

Get total.

Private Attributes

```
• double _total = 0
```

Total time.

clock_t _start

Start time.

5.105.1 Detailed Description

Stop watch.

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

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

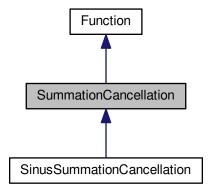
· lib/hnco/stop-watch.hh

5.106 SummationCancellation Class Reference

Summation cancellation.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SummationCancellation:



Public Member Functions

• SummationCancellation (int n)

Constructor.

size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &x)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Protected Member Functions

void convert (const bit_vector_t &x)
 Convert a bit vector into a real vector.

Protected Attributes

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

5.106.1 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.

5.106.2 Constructor & Destructor Documentation

5.106.2.1 SummationCancellation()

```
\begin{tabular}{lll} Summation Cancellation ( & & \\ & int n) & [inline] \end{tabular}
```

Constructor.

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

Parameters

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

Definition at line 71 of file cancellation.hh.

5.106.3 Member Function Documentation

5.106.3.1 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:

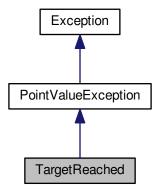
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.107 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

5.107.1 Detailed Description

target reached

Definition at line 62 of file exception.hh.

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

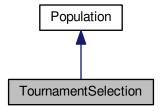
· lib/hnco/exception.hh

5.108 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

5.108.1 Detailed Description

Population with tournament selection.

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

5.108.2 Member Function Documentation

```
5.108.2.1 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:

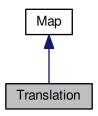
- · lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

5.109 Translation Class Reference

Translation.

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

Inheritance diagram for Translation:



Public Member Functions

• void random (int n)

Random instance.

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

Мар.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

5.109.1 Detailed Description

Translation.

A translation is an affine map f from \mathbb{Z}_2^n to itself defined by f(x)=x+b, where b is an n-dimensional bit vector. Definition at line 70 of file map.hh.

5.109.2 Member Function Documentation

```
5.109.2.1 is_surjective()
```

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 121 of file map.hh.

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

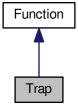
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.110 Trap Class Reference

Trap.

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

Inheritance diagram for Trap:



Public Member Functions

• Trap (int bv_size, int num_traps)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

• int _num_traps

Number of traps.

int _trap_size

Trap size.

5.110.1 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.

5.110.2 Constructor & Destructor Documentation

```
5.110.2.1 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.

5.110.3 Member Function Documentation

```
5.110.3.1 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.

5.110.3.2 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:

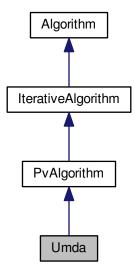
- · lib/hnco/functions/trap.hh
- lib/hnco/functions/trap.cc

5.111 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.

5.111.1 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:

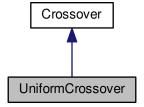
- lib/hnco/algorithms/pv/umda.hh
- lib/hnco/algorithms/pv/umda.cc

5.112 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.

5.112.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

5.112.2 Member Function Documentation

5.112.2.1 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:

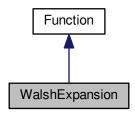
- · lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

5.113 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



Public Member Functions

• WalshExpansion ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

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.

Random instances

```
    template < class Generator >
        void random (int n, int num_features, Generator generator)
        Random instance.
```

• 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

5.113.1 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.

5.113.2 Member Function Documentation

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

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

Random instance.

Parameters

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

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

```
5.113.2.2 random() [2/2]
```

```
void random (
         int n,
         int num_features ) [inline]
```

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:

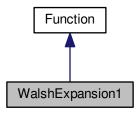
- · lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

5.114 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.

Random instances

 $\bullet \ \ \text{template}{<} \text{class Generator} >$

void random (int n, Generator generator)

Random instance.

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

• size_t get_bv_size ()

Get bit vector size.

double get_maximum ()

Get the global maximum.

bool has_known_maximum ()

Check for a known maximum.

bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Private Member Functions

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

Private Attributes

std::vector< double > _linear
 Linear part.

Friends

· class boost::serialization::access

5.114.1 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.

5.114.2 Member Function Documentation

5.114.2.1 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.

5.114.2.2 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.

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

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

Random instance.

Parameters

n	Size of bit vectors
generator	Weight generator

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

```
5.114.2.4 random() [2/2]  \begin{tabular}{ll} $\tt void\ random\ ($ & int\ n\ )$ & [inline] \end{tabular}
```

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:

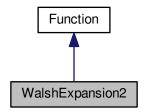
- · lib/hnco/functions/walsh/walsh-expansion-1.hh
- · lib/hnco/functions/walsh/walsh-expansion-1.cc

5.115 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

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

Inheritance diagram for WalshExpansion2:



Public Member Functions

• WalshExpansion2 ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Random instances

template < class LinearGen , class QuadraticGen > void random (int n, LinearGen linear_gen, QuadraticGen quadratic_gen)

Instance generators.

• void random (int n)

Random instance.

• 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.
- std::vector< std::vector< double >> _quadratic
 Quadratic part.

Friends

· class boost::serialization::access

5.115.1 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.

5.115.2 Member Function Documentation

5.115.2.1 generate_ising1_long_range()

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

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

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.

5.115.2.2 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

$$f(x) = \sum_{ij} J(d_{ij})(1 - 2x_i)(1 - 2x_j)$$

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, $d_{ij} = \min\{|i-j|, n-|i-j|\}$, 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.

5.115.2.3 random() [1/2]

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.

```
5.115.2.4 random() [2/2] void random ( int n ) [inline]
```

Random instance.

The weights are sampled from the normal distribution.

Parameters

```
n Size of bit vector
```

Definition at line 117 of file walsh-expansion-2.hh.

5.115.3 Member Data Documentation

```
5.115.3.1 _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

5.116 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.

5.116.1 Detailed Description

Walsh transform term.

Definition at line 46 of file function.hh.

5.116.2 Member Data Documentation

5.116.2.1 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::UniformCrossover, 233
hnco::algorithm::Population, 181	bv_from_vector_bool
_expression	hnco, 21
hnco::function::AbstractMaxSat, 29	bv_to_vector_bool
_functions	hnco, 21
hnco::algorithm::Algorithm, 37	
_lookup	Cache, 48
hnco::algorithm::Population, 181	hnco::function::Cache, 49
_q	CallCounter, 50
hnco::function::Qubo, 193	comma_selection
_quadratic	hnco::algorithm::Population, 177
hnco::function::WalshExpansion2, 242	CompactGa, 51
_second	CompleteSearch, 53
hnco::algorithm::hea::SpinMoment, 213	compute_walsh_transform
	hnco::function::Function, 69
AbstractLabs, 27	Crossover, 54
AbstractMaxSat, 28	December Lumps EE
AdditiveGaussianNoise, 30	DeceptiveJump, 55
AffineMap, 32	EqualProducts, 57
Algorithm, 34	Error, 59
	eval
bernoulli_trials	hnco::function::NearestNeighborIsingModel1, 140
hnco::neighborhood::MultiBitFlip, 134	hnco::function::NearestNeighborIsingModel2, 143
BernoulliProcess, 37	hnco::function::OnBudgetFunction, 159
hnco::neighborhood::BernoulliProcess, 38, 39	hnco::function::StopOnMaximum, 217
BiasedCrossover, 40	hnco::function::StopOnTarget, 220
bit_t	Exception, 61
hnco, 16	
BitHerding, 41	Factorization, 61
BitMoment, 43	hnco::function::Factorization, 63
bm_add_rows	feature
hnco, 17	hnco::function::Function::WalshTransformTerm,
bm_identity	243
hnco, 17	FirstAscentHillClimbing, 64
bm_invert	FourPeaks, 65
hnco, 18	Function, 68
bm_multiply	Function::WalshTransformTerm, 242
hnco, 18	FunctionController, 72
bm_rank	FunctionDecorator, 73
hnco, 19	FunctionMapComposition, 74
bm_row_echelon_form	hnco::function::FunctionMapComposition, 75
hnco, 19	FunctionModifier, 77
bm_solve	FunctionPlugin, 78
hnco, 19	hnco::function::FunctionPlugin, 79
bm_solve_upper_triangular	
hnco, 20	generate_ising1_long_range
BmPbil, 44	hnco::function::WalshExpansion2, 240
breed	generate_ising1_long_range_periodic
hnco::algorithm::BiasedCrossover, 40	hnco::function::WalshExpansion2, 241
hnco::algorithm::Crossover, 55	GeneticAlgorithm, 79

hnco::algorithm::GeneticAlgorithm, 81	hnco, 13
get_best_bv	bit_t, 16
hnco::algorithm::Population, 177, 178	bm_add_rows, 17
get_best_value	bm_identity, 17
hnco::algorithm::Population, 179	bm_invert, 18
get_last_improvement	bm_multiply, 18
hnco::function::ProgressTracker, 186	bm_rank, 19
get_maximum	bm_row_echelon_form, 19
hnco::function::AdditiveGaussianNoise, 31	bm_solve, 19
hnco::function::DeceptiveJump, 56	bm_solve_upper_triangular, 20
hnco::function::FourPeaks, 67	bv_from_vector_bool, 21
hnco::function::Function, 70	bv_to_vector_bool, 21
hnco::function::FunctionMapComposition, 76	perm_identity, 21
hnco::function::Hiff, 94	perm_random, 21
hnco::function::Jump, 103	sbm_multiply, 22
hnco::function::LeadingOnes, 107	sparse_bit_matrix_t, 17
hnco::function::LongPath, 117	sparse_bit_vector_t, 17
hnco::function::Needle, 146	hnco::AffineMap
hnco::function::Negation, 148	is_surjective, 33
hnco::function::OneMax, 162	random, 33
hnco::function::Plateau, 173	hnco::Injection
hnco::function::PriorNoise, 183	Injection, 97
hnco::function::Ridge, 201	hnco::LinearMap
hnco::function::SixPeaks, 209	is_surjective, 112
hnco::function::Trap, 230	random, 113
get_worst_bv	hnco::Map
hnco::algorithm::Population, 179, 180	is_surjective, 119
HammingBall, 82	hnco::MapComposition
hnco::neighborhood::HammingBall, 83	is_surjective, 121
HammingSphere, 84	MapComposition, 121
hnco::neighborhood::HammingSphere, 85	hnco::Permutation
HammingSphereIterator, 85	is_surjective, 171
hnco::neighborhood::HammingSphereIterator, 87	hnco::Projection
has_known_maximum	is_surjective, 189
hnco::function::AdditiveGaussianNoise, 31	Projection, 188
hnco::function::DeceptiveJump, 56	hnco::Translation
hnco::function::FourPeaks, 67	is_surjective, 228
hnco::function::FunctionMapComposition, 76	hnco::algorithm, 22
hnco::function::Hiff, 94	hnco::algorithm::Algorithm
hnco::function::Jump, 103	_functions, 37
hnco::function::LeadingOnes, 107	set_solution, 36
hnco::function::LinearFunction, 110	update_solution, 36
hnco::function::LongPath, 117	hnco::algorithm::BiasedCrossover
hnco::function::Needle, 146	breed, 40
hnco::function::Negation, 148	hnco::algorithm::Crossover
hnco::function::OneMax, 162	breed, 55
hnco::function::Plateau, 173	hnco::algorithm::GeneticAlgorithm
hnco::function::PriorNoise, 183	GeneticAlgorithm, 81
hnco::function::Ridge, 201	set_allow_stay, 82
hnco::function::SixPeaks, 209	hnco::algorithm::IterativeAlgorithm
hnco::function::SummationCancellation, 224	IterativeAlgorithm, 99
hnco::function::Trap, 230	maximize, 100
hnco::function::WalshExpansion1, 237	set_num_iterations, 100
Hboa, 87	hnco::algorithm::MuCommaLambdaEa
Hea	MuCommaLambdaEa, 132
hnco::algorithm::hea::Hea, 91	set_allow_stay, 133
Hea< Moment, Herding >, 88	hnco::algorithm::MuPlusLambdaEa
Hiff, 93	MuPlusLambdaEa, 137

set_allow_stay, 137	get_maximum, 70
hnco::algorithm::OnePlusLambdaCommaLambdaGa	incremental_eval, 70
OnePlusLambdaCommaLambdaGa, 164	provides_incremental_evaluation, 71
hnco::algorithm::OnePlusOneEa	safe eval, 71
OnePlusOneEa, 166	hnco::function::WalshTransformTerm
set_allow_stay, 167	feature, 243
set_num_iterations, 167	hnco::function::FunctionController
hnco::algorithm::Population	provides_incremental_evaluation, 73
compare index value, 181	hnco::function::FunctionMapComposition
lookup, 181	FunctionMapComposition, 75
comma_selection, 177	get_maximum, 76
get best bv, 177, 178	has_known_maximum, 76
get_best_value, 179	hnco::function::FunctionPlugin
get_worst_bv, 179, 180	FunctionPlugin, 79
plus_selection, 180	hnco::function::Hiff
hnco::algorithm::RandomLocalSearch	get_maximum, 94
set_patience, 195	has_known_maximum, 94
hnco::algorithm::SimulatedAnnealing	hnco::function::Jump
init beta, 203	get maximum, 103
hnco::algorithm::TournamentSelection	has_known_maximum, 103
select, 226	hnco::function::LeadingOnes
hnco::algorithm::UniformCrossover	get_maximum, 107
breed, 233	has known maximum, 107
hnco::algorithm::bm pbil, 25	hnco::function::LinearFunction
hnco::algorithm::bm_pbil::BmPbil	has_known_maximum, 110
set_selection_size, 47	provides_incremental_evaluation, 110
hnco::algorithm::hea, 25	random, 110, 111
hnco::algorithm::hea::Hea	hnco::function::LongPath
Hea, 91	get_maximum, 117
set_reset_period, 92	has_known_maximum, 117
set_selection_size, 92	hnco::function::MaxNae3Sat
hnco::algorithm::hea::SpinHerding	load, 123
q_variation, 212	hnco::function::MaxSat
SpinHerding, 212	random, 125
hnco::algorithm::hea::SpinMoment	hnco::function::NearestNeighborIsingModel1
_second, 213	eval, 140
hnco::function::AbstractMaxSat	provides_incremental_evaluation, 140
_expression, 29	random, 140, 141
load, 29	hnco::function::NearestNeighborIsingModel2
hnco::function::AdditiveGaussianNoise	eval, 143
get_maximum, 31	provides_incremental_evaluation, 143
has_known_maximum, 31	random, 144
hnco::function::Cache	hnco::function::Needle
Cache, 49	get_maximum, 146
provides_incremental_evaluation, 49	has_known_maximum, 146
hnco::function::DeceptiveJump	hnco::function::Negation
get_maximum, 56	get_maximum, 148
has_known_maximum, 56	has_known_maximum, 148
hnco::function::EqualProducts	provides_incremental_evaluation, 148
random, 58, 59	hnco::function::NkLandscape
hnco::function::Factorization	random, 155
Factorization, 63	random_structure, 156
load, 63	hnco::function::OnBudgetFunction
hnco::function::FourPeaks	eval, 159
get_maximum, 67	incremental_eval, 159
has_known_maximum, 67	update, 160
hnco::function::Function	hnco::function::OneMax
compute_walsh_transform, 69	get_maximum, 162

has_known_maximum, 162	HammingSphereIterator, 87
provides_incremental_evaluation, 162	hnco::neighborhood::MultiBitFlip
hnco::function::Plateau	bernoulli_trials, 134
get_maximum, 173	MultiBitFlip, 134
has_known_maximum, 173	reservoir_sampling, 135
hnco::function::PriorNoise	hnco::neighborhood::Neighborhood
get_maximum, 183	map, 151
has_known_maximum, 183	mutate, 152
provides_incremental_evaluation, 183	Neighborhood, 151
hnco::function::ProgressTracker	hnco::neighborhood::NeighborhoodIterator
get_last_improvement, 186	NeighborhoodIterator, 153
hnco::function::Qubo	hnco::neighborhood::SingleBitFlipIterator
_q, 193	SingleBitFlipIterator, 206
load, 192	hnco::random, 26
hnco::function::Ridge	HncoEvaluator, 94
get_maximum, 201	Hypercubelterator, 95
has_known_maximum, 201	
hnco::function::SixPeaks	incremental_eval
get_maximum, 209	hnco::function::Function, 70
has_known_maximum, 209	hnco::function::OnBudgetFunction, 159
hnco::function::StopOnMaximum	hnco::function::StopOnMaximum, 217
eval, 217	hnco::function::StopOnTarget, 220
incremental_eval, 217	init_beta
StopOnMaximum, 216	hnco::algorithm::SimulatedAnnealing, 203
update, 217	Injection, 96
hnco::function::StopOnTarget	hnco::Injection, 97
eval, 220	is_surjective
incremental_eval, 220	hnco::AffineMap, 33
StopOnTarget, 219	hnco::LinearMap, 112
update, 220	hnco::Map, 119
hnco::function::SummationCancellation	hnco::MapComposition, 121
has_known_maximum, 224	hnco::Permutation, 171
SummationCancellation, 223	hnco::Projection, 189
hnco::function::Trap	hnco::Translation, 228
get_maximum, 230	IterativeAlgorithm, 98
has_known_maximum, 230	hnco::algorithm::IterativeAlgorithm, 99
Trap, 229	Iterator, 101
hnco::function::WalshExpansion	
random, 235	Jump, 102
hnco::function::WalshExpansion1	
has_known_maximum, 237	Labs, 104
provides_incremental_evaluation, 237	LabsMeritFactor, 105
random, 238	LastEvaluation, 106
hnco::function::WalshExpansion2	LeadingOnes, 106
_quadratic, 242	LinearFunction, 108
generate_ising1_long_range, 240	LinearMap, 111
	load
generate_ising1_long_range_periodic, 241 random, 241, 242	hnco::function::AbstractMaxSat, 29
hnco::neighborhood, 26	hnco::function::Factorization, 63
_	hnco::function::MaxNae3Sat, 123
hnco::neighborhood::BernoulliProcess	hnco::function::Qubo, 192
BernoulliProcess, 38, 39	LocalMaximum, 113
set_allow_stay, 39	LogContext, 114
set_probability, 39	LongPath, 115
hnco::neighborhood::HammingBall	Ltga, 118
HammingBall, 83	Man. 440
hnco::neighborhood::HammingSphere	Map, 119
HammingSphere, 85	map
hnco::neighborhood::HammingSphereIterator	hnco::neighborhood::Neighborhood, 151

MapComposition, 120 hnco::MapComposition, 121 MaxNae3Sat, 123 MaxSat, 124 maximize hnco::algorithm::IterativeAlgorithm, 100 MaximumReached, 122 Mimic, 126	hnco::function::Function, 71 hnco::function::FunctionController, 73 hnco::function::LinearFunction, 110 hnco::function::NearestNeighborIsingModel1, 140 hnco::function::NearestNeighborIsingModel2, 143 hnco::function::Negation, 148 hnco::function::OneMax, 162 hnco::function::PriorNoise, 183
Mmas, 128 Model, 129 ModelParameters, 130	hnco::function::WalshExpansion1, 237 PvAlgorithm, 189
MuCommaLambdaEa, 131 hnco::algorithm::MuCommaLambdaEa, 132 MuPlusLambdaEa, 135	q_variation hnco::algorithm::hea::SpinHerding, 212 Qubo, 191
hnco::algorithm::MuPlusLambdaEa, 137 MultiBitFlip, 133 hnco::neighborhood::MultiBitFlip, 134	Random, 193 random
mutate hnco::neighborhood::Neighborhood, 152	hnco::AffineMap, 33 hnco::LinearMap, 113 hnco::function::EqualProducts, 58, 59
NearestNeighborIsingModel1, 138 NearestNeighborIsingModel2, 141 Neadle 145	hnco::function::LinearFunction, 110, 111 hnco::function::MaxSat, 125 hnco::function::NearestNeighborIsingModel1, 140,
Needle, 145 Negation, 147 Neighborhood, 149	141 hnco::function::NearestNeighborlsingModel2, 144
hnco::neighborhood::Neighborhood, 151 NeighborhoodIterator, 152 hnco::neighborhood::NeighborhoodIterator, 153	hnco::function::NkLandscape, 155 hnco::function::WalshExpansion, 235 hnco::function::WalshExpansion1, 238
NkLandscape, 153 NpsPbil, 156	hnco::function::WalshExpansion2, 241, 242 random_structure hnco::function::NkLandscape, 156
OnBudgetFunction, 158 OneMax, 160	RandomLocalSearch, 194 RandomSearch, 196
OnePlusLambdaCommaLambdaGa, 163 hnco::algorithm::OnePlusLambdaComma← LambdaGa, 164	RandomWalk, 197 reservoir_sampling hnco::neighborhood::MultiBitFlip, 135
OnePlusOneEa, 165 hnco::algorithm::OnePlusOneEa, 166	Restart, 198 Ridge, 200
ParameterLessPopulationPyramid, 167 Pbil, 169	safe_eval hnco::function::Function, 71
perm_identity hnco, 21 perm_random	sbm_multiply hnco, 22 select
hnco, 21 Permutation, 170	hnco::algorithm::TournamentSelection, 226 set_allow_stay
Plateau, 172 plus_selection hnco::algorithm::Population, 180	hnco::algorithm::GeneticAlgorithm, 82 hnco::algorithm::MuCommaLambdaEa, 133 hnco::algorithm::MuPlusLambdaEa, 137
PointValueException, 174 Population, 175 PriorNoise, 182	hnco::algorithm::OnePlusOneEa, 167 hnco::neighborhood::BernoulliProcess, 39 set_num_iterations
ProgressTracker, 184 ProgressTracker::Event, 60	hnco::algorithm::IterativeAlgorithm, 100 hnco::algorithm::OnePlusOneEa, 167
ProgressTrackerContext, 186 Projection, 187 hnco::Projection, 188	set_patience hnco::algorithm::RandomLocalSearch, 195 set_probability
provides_incremental_evaluation hnco::function::Cache, 49	hnco::neighborhood::BernoulliProcess, 39 set_reset_period

```
hnco::algorithm::hea::Hea, 92
set selection size
    hnco::algorithm::bm_pbil::BmPbil, 47
    hnco::algorithm::hea::Hea, 92
set_solution
    hnco::algorithm::Algorithm, 36
SimulatedAnnealing, 202
SingleBitFlip, 204
SingleBitFlipIterator, 205
     hnco::neighborhood::SingleBitFlipIterator, 206
SinusSummationCancellation, 206
SixPeaks, 207
sparse_bit_matrix_t
    hnco, 17
sparse_bit_vector_t
    hnco, 17
SpinHerding, 209
    hnco::algorithm::hea::SpinHerding, 212
SpinMoment, 212
SteepestAscentHillClimbing, 214
StopOnMaximum, 215
     hnco::function::StopOnMaximum, 216
StopOnTarget, 218
    hnco::function::StopOnTarget, 219
StopWatch, 221
SummationCancellation, 222
     hnco::function::SummationCancellation, 223
TargetReached, 224
TournamentSelection, 225
Translation, 227
Trap, 228
     hnco::function::Trap, 229
Umda, 231
UniformCrossover, 232
update
     hnco::function::OnBudgetFunction, 160
     hnco::function::StopOnMaximum, 217
    hnco::function::StopOnTarget, 220
update_solution
    hnco::algorithm::Algorithm, 36
WalshExpansion, 233
WalshExpansion1, 236
WalshExpansion2, 239
```