HNCO

0.9

Generated by Doxygen 1.8.13

Contents

1	Nam	espace	Index		1
	1.1	Names	space List		. 1
2	Hier	archical	Index		3
	2.1	Class I	Hierarchy		. 3
3	Clas	s Index			7
	3.1	Class I	_ist		. 7
4	Nam	nespace	Docume	ntation	11
	4.1	hnco N	lamespace	e Reference	. 11
		4.1.1	Detailed	Description	. 14
		4.1.2	Typedef	Documentation	. 14
			4.1.2.1	bit_t	. 14
			4.1.2.2	sparse_bit_matrix_t	. 15
			4.1.2.3	sparse_bit_vector_t	. 15
		4.1.3	Function	Documentation	. 15
			4.1.3.1	bm_add_rows()	. 15
			4.1.3.2	bm_identity()	. 16
			4.1.3.3	bm_invert()	. 16
			4.1.3.4	bm_multiply()	. 17
			4.1.3.5	bm_rank()	. 17
			4.1.3.6	bm_row_echelon_form()	. 17
			4.1.3.7	bm_solve()	. 17
			4.1.3.8	bm solve upper triangular()	. 18

ii CONTENTS

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

CONTENTS

	5.3.2	Member Function Documentation	33
		5.3.2.1 set_solution()	33
		5.3.2.2 update_solution()	33
	5.3.3	Member Data Documentation	33
		5.3.3.1 _functions	34
5.4	Bernou	ulliProcess Class Reference	34
	5.4.1	Detailed Description	35
	5.4.2	Constructor & Destructor Documentation	35
		5.4.2.1 BernoulliProcess() [1/2]	35
		5.4.2.2 BernoulliProcess() [2/2]	36
	5.4.3	Member Function Documentation	36
		5.4.3.1 set_allow_stay()	36
		5.4.3.2 set_probability()	36
5.5	Biased	Crossover Class Reference	37
	5.5.1	Detailed Description	37
	5.5.2	Member Function Documentation	37
		5.5.2.1 breed()	37
5.6	Binary	Herding Class Reference	38
	5.6.1	Detailed Description	39
	5.6.2	Member Enumeration Documentation	39
		5.6.2.1 anonymous enum	39
5.7	Binary	Moment Struct Reference	40
	5.7.1	Detailed Description	41
5.8	BmPbi	I Class Reference	41
	5.8.1	Detailed Description	43
	5.8.2	Member Enumeration Documentation	43
		5.8.2.1 anonymous enum	43
		5.8.2.2 anonymous enum	44
		5.8.2.3 anonymous enum	44
	5.8.3	Member Function Documentation	44

iv CONTENTS

		5.8.3.1 set_selection_size()	44
5.9	Cache	Class Reference	45
	5.9.1	Detailed Description	46
	5.9.2	Constructor & Destructor Documentation	46
		5.9.2.1 Cache()	46
	5.9.3	Member Function Documentation	46
		5.9.3.1 provides_incremental_evaluation()	47
5.10	CallCo	unter Class Reference	47
	5.10.1	Detailed Description	48
5.11	Compa	actGa Class Reference	48
	5.11.1	Detailed Description	50
5.12	Comple	eteSearch Class Reference	50
	5.12.1	Detailed Description	51
5.13	Crosso	ver Class Reference	51
	5.13.1	Detailed Description	51
	5.13.2	Member Function Documentation	52
		5.13.2.1 breed()	52
5.14	Decept	tiveJump Class Reference	52
	5.14.1	Detailed Description	53
	5.14.2	Member Function Documentation	53
		5.14.2.1 get_maximum()	53
		5.14.2.2 has_known_maximum()	54
5.15	EqualP	Products Class Reference	54
	5.15.1	Detailed Description	55
	5.15.2	Member Function Documentation	55
		5.15.2.1 random()	55
5.16	Error C	Class Reference	56
	5.16.1	Detailed Description	57
5.17	Progre	ssTracker::Event Struct Reference	57
	5.17.1	Detailed Description	57

CONTENTS

5.18	Except	ion Class Reference	57
	5.18.1	Detailed Description	58
5.19	Factoria	zation Class Reference	58
	5.19.1	Detailed Description	59
	5.19.2	Constructor & Destructor Documentation	59
		5.19.2.1 Factorization()	59
5.20	FirstAs	centHillClimbing Class Reference	60
	5.20.1	Detailed Description	61
5.21	FourPe	eaks Class Reference	61
	5.21.1	Detailed Description	62
	5.21.2	Member Function Documentation	63
		5.21.2.1 get_maximum()	63
		5.21.2.2 has_known_maximum()	63
5.22	Functio	on Class Reference	64
	5.22.1	Detailed Description	65
	5.22.2	Member Function Documentation	65
		5.22.2.1 compute_walsh_transform()	65
		5.22.2.2 get_maximum()	66
		5.22.2.3 incremental_eval()	66
		5.22.2.4 provides_incremental_evaluation()	67
		5.22.2.5 safe_eval()	67
5.23	Functio	onController Class Reference	68
	5.23.1	Detailed Description	69
	5.23.2	Member Function Documentation	69
		5.23.2.1 provides_incremental_evaluation()	69
5.24	Function	onDecorator Class Reference	69
	5.24.1	Detailed Description	70
5.25	Function	onMapComposition Class Reference	70
		Detailed Description	71
	5.25.2	Constructor & Destructor Documentation	71

vi

		5.25.2.1 FunctionMapComposition()	71
	5.25.3	Member Function Documentation	72
		5.25.3.1 get_maximum()	72
		5.25.3.2 has_known_maximum()	72
5.26	Function	onModifier Class Reference	73
	5.26.1	Detailed Description	73
5.27	Function	onPlugin Class Reference	74
	5.27.1	Detailed Description	75
	5.27.2	Constructor & Destructor Documentation	75
		5.27.2.1 FunctionPlugin()	75
5.28	Genetic	cAlgorithm Class Reference	75
	5.28.1	Detailed Description	77
	5.28.2	Constructor & Destructor Documentation	77
		5.28.2.1 GeneticAlgorithm()	77
	5.28.3	Member Function Documentation	78
		5.28.3.1 set_allow_stay()	78
5.29	Hammi	ingBall Class Reference	78
	5.29.1	Detailed Description	79
	5.29.2	Constructor & Destructor Documentation	79
		5.29.2.1 HammingBall()	79
5.30	Hammi	ingSphere Class Reference	80
	5.30.1	Detailed Description	81
	5.30.2	Constructor & Destructor Documentation	81
		5.30.2.1 HammingSphere()	81
5.31	Hammi	ingSphereIterator Class Reference	81
	5.31.1	Detailed Description	82
	5.31.2	Constructor & Destructor Documentation	83
		5.31.2.1 HammingSphereIterator()	83
5.32	Hboa C	Class Reference	83
	5.32.1	Detailed Description	84

CONTENTS vii

5.33	Hea<	Moment, Herding > Class Template Reference	84
	5.33.1	Detailed Description	87
	5.33.2	Member Enumeration Documentation	87
		5.33.2.1 anonymous enum	87
		5.33.2.2 anonymous enum	87
	5.33.3	Constructor & Destructor Documentation	88
		5.33.3.1 Hea()	88
	5.33.4	Member Function Documentation	88
		5.33.4.1 set_reset_period()	88
		5.33.4.2 set_selection_size()	88
5.34	Hiff Cla	ass Reference	89
	5.34.1	Detailed Description	90
	5.34.2	Member Function Documentation	90
		5.34.2.1 get_maximum()	90
		5.34.2.2 has_known_maximum()	90
5.35	HncoE	valuator Class Reference	91
	5.35.1	Detailed Description	91
5.36	Hypero	cubelterator Class Reference	92
	5.36.1	Detailed Description	92
5.37	Iterativ	eAlgorithm Class Reference	93
	5.37.1	Detailed Description	94
	5.37.2	Constructor & Destructor Documentation	94
		5.37.2.1 IterativeAlgorithm()	94
	5.37.3	Member Function Documentation	94
		5.37.3.1 maximize()	94
		5.37.3.2 set_num_iterations()	95
5.38	Iterator	Class Reference	95
	5.38.1	Detailed Description	96
5.39	Jump (Class Reference	97
	5.39.1	Detailed Description	97

viii CONTENTS

	5.39.2	Member Function Documentation	 	 98
		5.39.2.1 get_maximum()	 	 98
		5.39.2.2 has_known_maximum()	 	 98
5.40	Labs C	lass Reference	 	 98
	5.40.1	Detailed Description	 	 99
5.41	LastEv	aluation Class Reference	 	 99
	5.41.1	Detailed Description	 	 100
5.42	Leading	gOnes Class Reference	 	 100
	5.42.1	Detailed Description	 	 101
	5.42.2	Member Function Documentation	 	 101
		5.42.2.1 get_maximum()	 	 101
		5.42.2.2 has_known_maximum()	 	 101
5.43	LinearF	function Class Reference	 	 102
	5.43.1	Detailed Description	 	 103
	5.43.2	Member Function Documentation	 	 103
		5.43.2.1 has_known_maximum()	 	 103
		5.43.2.2 random()	 	 103
5.44	Linear	Map Class Reference	 	 104
	5.44.1	Detailed Description	 	 105
	5.44.2	Member Function Documentation	 	 105
		5.44.2.1 is_surjective()	 	 105
		5.44.2.2 random()	 	 105
5.45	LocalM	aximum Class Reference	 	 106
	5.45.1	Detailed Description	 	 106
5.46	LongPa	th Class Reference	 	 107
	5.46.1	Detailed Description	 	 107
5.47	Ltga Cl	ass Reference	 	 108
	5.47.1	Detailed Description	 	 109
5.48	Map Cl	ass Reference	 	 109
	5.48.1	Detailed Description	 	 110

CONTENTS

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

CONTENTS

5.56	MultiBit	Flip Class Reference	121
	5.56.1	Detailed Description	122
	5.56.2	Constructor & Destructor Documentation	122
		5.56.2.1 MultiBitFlip()	122
	5.56.3	Member Function Documentation	122
		5.56.3.1 bernoulli_trials()	122
		5.56.3.2 reservoir_sampling()	123
5.57	MuPlus	sLambdaEa Class Reference	123
	5.57.1	Detailed Description	124
	5.57.2	Constructor & Destructor Documentation	125
		5.57.2.1 MuPlusLambdaEa()	125
	5.57.3	Member Function Documentation	125
		5.57.3.1 set_allow_stay()	125
5.58	Needle	Class Reference	126
	5.58.1	Detailed Description	126
	5.58.2	Member Function Documentation	127
		5.58.2.1 get_maximum()	127
		5.58.2.2 has_known_maximum()	127
5.59	Negatio	on Class Reference	128
	5.59.1	Detailed Description	129
	5.59.2	Member Function Documentation	129
		5.59.2.1 get_maximum()	129
		5.59.2.2 has_known_maximum()	129
		5.59.2.3 provides_incremental_evaluation()	130
5.60	Neighb	orhood Class Reference	130
	5.60.1	Detailed Description	132
	5.60.2	Constructor & Destructor Documentation	132
		5.60.2.1 Neighborhood()	132
	5.60.3	Member Function Documentation	132
		5.60.3.1 map()	132

CONTENTS xi

		5.60.3.2 mutate()	133
5.61	Neighb	orhoodIterator Class Reference	133
	5.61.1	Detailed Description	134
	5.61.2	Constructor & Destructor Documentation	134
		5.61.2.1 NeighborhoodIterator()	134
5.62	NkLand	dscape Class Reference	134
	5.62.1	Detailed Description	135
	5.62.2	Member Function Documentation	136
		5.62.2.1 random()	136
5.63	NpsPb	il Class Reference	136
	5.63.1	Detailed Description	138
5.64	OnBud	getFunction Class Reference	138
	5.64.1	Detailed Description	140
	5.64.2	Member Function Documentation	140
		5.64.2.1 eval()	140
		5.64.2.2 incremental_eval()	140
		5.64.2.3 update()	141
5.65	OneMa	ax Class Reference	141
	5.65.1	Detailed Description	142
	5.65.2	Member Function Documentation	142
		5.65.2.1 get_maximum()	142
		5.65.2.2 has_known_maximum()	143
		5.65.2.3 provides_incremental_evaluation()	143
5.66	OnePlu	usLambdaCommaLambdaGa Class Reference	143
	5.66.1	Detailed Description	144
	5.66.2	Constructor & Destructor Documentation	145
		5.66.2.1 OnePlusLambdaCommaLambdaGa()	145
5.67	OnePlu	usOneEa Class Reference	145
	5.67.1	Detailed Description	146
	5.67.2	Constructor & Destructor Documentation	147

xii CONTENTS

		5.67.2.1 OnePlusOneEa()
	5.67.3	Member Function Documentation
		5.67.3.1 set_allow_stay()
		5.67.3.2 set_num_iterations()
5.68	Parame	eterLessPopulationPyramid Class Reference
	5.68.1	Detailed Description
5.69	Pbil Cla	ass Reference
	5.69.1	Detailed Description
5.70	Permut	tation Class Reference
	5.70.1	Detailed Description
	5.70.2	Member Function Documentation
		5.70.2.1 is_surjective()
5.71	Plateau	u Class Reference
	5.71.1	Detailed Description
	5.71.2	Member Function Documentation
		5.71.2.1 get_maximum()
		5.71.2.2 has_known_maximum()
5.72	PointVa	alueException Class Reference
	5.72.1	Detailed Description
5.73	Popula	tion Class Reference
	5.73.1	Detailed Description
	5.73.2	Member Function Documentation
		5.73.2.1 comma_selection()
		5.73.2.2 get_best_bv() [1/2]
		5.73.2.3 get_best_bv() [2/2]
		5.73.2.4 get_best_value() [1/2]
		5.73.2.5 get_best_value() [2/2]
		5.73.2.6 get_worst_bv()
		5.73.2.7 plus_selection()
	5.73.3	Member Data Documentation

CONTENTS xiii

		5.73.3.1 _co	mpare_index_	_value		 	 	 		159
		5.73.3.2 _lo	okup			 	 	 		159
5.74	PriorNo	oise Class Refe	rence			 	 	 		160
	5.74.1	Detailed Desc	ription			 	 	 		161
	5.74.2	Member Fund	tion Documen	tation		 	 	 		161
		5.74.2.1 get	_maximum()			 	 	 		161
		5.74.2.2 has	_known_maxi	mum()		 	 	 		161
		5.74.2.3 pro	vides_increme	ental_evalua	ation()	 	 	 		162
5.75	Progre	ssTracker Clas	Reference .			 	 	 		162
	5.75.1	Detailed Desc	ription			 	 	 		163
	5.75.2	Member Fund	tion Documen	tation		 	 	 		164
		5.75.2.1 eva	l()			 	 	 		164
		5.75.2.2 get	_last_improve	ment()		 	 	 		164
		5.75.2.3 incr	emental_eval()		 	 	 		164
		5.75.2.4 upo	ate()			 	 	 		165
5.76	PvAlgo	rithm Class Re	ference			 	 	 		165
	5.76.1	Detailed Desc	ription			 	 	 		166
	5.76.2	Member Enur	neration Docu	mentation		 	 	 		166
		5.76.2.1 and	nymous enum	١		 	 	 		166
5.77	Qubo (Class Referenc	e			 	 	 		167
	5.77.1	Detailed Desc	ription			 	 	 		168
	5.77.2	Member Fund	tion Documen	tation		 	 	 		168
		5.77.2.1 load	d()			 	 	 		168
	5.77.3	Member Data	Documentation	on		 	 	 		169
		5.77.3.1 _q				 	 	 		169
5.78	Rando	m Struct Refere	ence			 	 	 		169
	5.78.1	Detailed Desc	ription			 	 	 		169
5.79	Rando	mLocalSearch	Class Referen	ce		 	 	 		170
	5.79.1	Detailed Desc	ription			 	 	 		171
	5.79.2	Member Fund	tion Documen	tation		 	 	 		171

xiv CONTENTS

		5.79.2.1	set_patiend	e()			 	 	 	 	 171
5.80	Rando	mSearch C	Class Refere	nce			 	 	 	 	 172
	5.80.1	Detailed I	Description				 	 	 	 	 173
5.81	Restart	t Class Re	ference				 	 	 	 	 173
	5.81.1	Detailed I	Description				 	 	 	 	 174
5.82	Ridge (Class Refe	erence				 	 	 	 	 174
	5.82.1	Detailed I	Description				 	 	 	 	 175
	5.82.2	Member I	Function Do	cumentatio	on		 	 	 	 	 175
		5.82.2.1	get_maxim	um()			 	 	 	 	 175
		5.82.2.2	has_known	_maximur	n()		 	 	 	 	 176
5.83	Simula	tedAnneali	ing Class Re	eference .			 	 	 	 	 176
	5.83.1	Detailed I	Description				 	 	 	 	 178
	5.83.2	Member I	Function Do	cumentatio	on		 	 	 	 	 178
		5.83.2.1	init_beta()				 	 	 	 	 178
5.84	Single	BitFlip Clas	ss Reference)			 	 	 	 	 178
	5.84.1	Detailed I	Description				 	 	 	 	 179
5.85	Single	BitFlipItera	tor Class Re	ference .			 	 	 	 	 179
	5.85.1	Detailed I	Description				 	 	 	 	 180
	5.85.2	Construc	tor & Destru	ctor Docur	nentatio	on	 	 	 	 	 180
		5.85.2.1	SingleBitFli	plterator()			 	 	 	 	 180
5.86	SinusS	ummation	Cancellation	Class Ref	ference		 	 	 	 	 181
	5.86.1	Detailed I	Description				 	 	 	 	 181
5.87	SixPea	ks Class F	Reference .				 	 	 	 	 182
	5.87.1	Detailed I	Description				 	 	 	 	 183
	5.87.2	Member I	Function Do	cumentatio	on		 	 	 	 	 183
		5.87.2.1	get_maxim	um()			 	 	 	 	 183
		5.87.2.2	has_known	_maximur	n()		 	 	 	 	 184
5.88	SpinHe	erding Clas	s Reference				 	 	 	 	 184
	5.88.1	Detailed I	Description				 	 	 	 	 185
	5.88.2	Member I	Enumeration	Documer	ntation		 	 	 	 	 186

CONTENTS xv

		5.88.2.1 anonymous enum	86
	5.88.3	Constructor & Destructor Documentation	86
		5.88.3.1 SpinHerding()	86
	5.88.4	Member Function Documentation	86
		5.88.4.1 q_variation()	86
5.89	SpinMo	oment Struct Reference	87
	5.89.1	Detailed Description	88
5.90	Steepe	stAscentHillClimbing Class Reference	88
	5.90.1	Detailed Description	89
5.91	StopOr	nMaximum Class Reference	89
	5.91.1	Detailed Description	90
	5.91.2	Constructor & Destructor Documentation	90
		5.91.2.1 StopOnMaximum()	90
	5.91.3	Member Function Documentation	91
		5.91.3.1 eval()	91
		5.91.3.2 incremental_eval()	91
		5.91.3.3 update()	91
5.92	StopOr	Target Class Reference	92
	5.92.1	Detailed Description	93
	5.92.2	Constructor & Destructor Documentation	93
		5.92.2.1 StopOnTarget()	93
	5.92.3	Member Function Documentation	93
		5.92.3.1 eval()	93
		5.92.3.2 incremental_eval()	94
		5.92.3.3 update()	94
5.93	Summa	ationCancellation Class Reference	94
	5.93.1	Detailed Description	96
	5.93.2	Constructor & Destructor Documentation	96
		5.93.2.1 SummationCancellation()	96
	5.93.3	Member Function Documentation	96

xvi CONTENTS

5.93.3.1 has_known_maximum()	196
5.94 TargetReached Class Reference	197
5.94.1 Detailed Description	197
5.95 TournamentSelection Class Reference	198
5.95.1 Detailed Description	198
5.95.2 Member Function Documentation	199
5.95.2.1 select()	199
5.96 Translation Class Reference	199
5.96.1 Detailed Description	200
5.96.2 Member Function Documentation	200
5.96.2.1 is_surjective()	201
5.97 Trap Class Reference	201
5.97.1 Detailed Description	202
5.97.2 Constructor & Destructor Documentation	202
5.97.2.1 Trap()	202
5.97.3 Member Function Documentation	202
5.97.3.1 get_maximum()	203
5.97.3.2 has_known_maximum()	203
5.98 Umda Class Reference	203
5.98.1 Detailed Description	205
5.99 UniformCrossover Class Reference	205
5.99.1 Detailed Description	205
5.99.2 Member Function Documentation	206
5.99.2.1 breed()	206
5.100WalshExpansion Class Reference	206
5.100.1 Detailed Description	207
5.100.2 Member Function Documentation	207
5.100.2.1 random()	207
5.101WalshExpansion1 Class Reference	208
5.101.1 Detailed Description	209
5.101.2 Member Function Documentation	209
5.101.2.1 random()	209
5.102WalshExpansion2 Class Reference	210
5.102.1 Detailed Description	211
5.102.2 Member Function Documentation	211
5.102.2.1 random()	211
5.102.3 Member Data Documentation	211
5.102.3.1 _quadratic	211
5.103Function::WalshTransformTerm Struct Reference	212
5.103.1 Detailed Description	212
5.103.2 Member Data Documentation	212
5.103.2.1 feature	212

CONTENTS	xvii
Index	213

Chapter 1

Namespace Index

1.1 Namespace List

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

nnco	
Top-level HNCO namespace	11
hnco::algorithm	
Algorithms	20
hnco::algorithm::bm_pbil	
Boltzmann machine PBIL	22
hnco::algorithm::hea	
Herding evolutionary algorithm	22
hnco::exception	
Exceptions	23
hnco::function	
Functions to be maximized	23
hnco::neighborhood	
Neighborhoods for local search	25
hnco::random	
Pseudo random numbers	26

2 Namespace Index

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Algorithm	31
CompleteSearch	50
Hboa	83
Ltga	
ParameterLessPopulationPyramid	
IterativeAlgorithm	
BmPbil	
FirstAscentHillClimbing	
GeneticAlgorithm	
Hea < Moment, Herding >	
MuCommaLambdaEa	
MuPlusLambdaEa	
OnePlusLambdaCommaLambdaGa	
PvAlgorithm	
CompactGa	
Mmas	
NpsPbil	
Pbil	
Umda	
RandomLocalSearch	
Restart	
SimulatedAnnealing	
SteepestAscentHillClimbing	
OnePlusOneEa	
, ,	38
-	40
	51
BiasedCrossover	
UniformCrossover	05
Evaluator	
HncoEvaluator	
· · · · · · · · · · · · · · · · · · ·	57
Exception	57

4 Hierarchical Index

Error	99
LocalMaximum	
MaximumReached	
TargetReached	197
Function	64
DeceptiveJump	
EqualProducts	
Factorization	
FunctionDecorator	
FunctionController	
Cache	
CallCounter	47
OnBudgetFunction	
ProgressTracker	
StopOnMaximum	
StopOnTarget	
AdditiveGaussianNoise	
FunctionMapComposition	
Negation	
PriorNoise	160
FunctionPlugin	74
Hiff	
Jump	
Labs	
LinearFunction	
LongPath	
MaxSat	
Needle	
NkLandscape	
OneMax	
Plateau	
Ridge	
SixPeaks	
SummationCancellation	194
SinusSummationCancellation	
Trap	
Walsh Expansion	
WalshExpansion1	
Iterator	
Hypercubelterator	
NeighborhoodIterator	
HammingSphereIterator	
SingleBitFlipIterator	
Map	109
AffineMap	29
LinearMap	
MapComposition	
Permutation	
Translation	
Model	117

2.1 Class Hierarchy 5

ModelParameters	
Neighborhood	0
MultiBitFlip	!1
BernoulliProcess	4
HammingBall	
HammingSphere	
SingleBitFlip	8
Population	5
TournamentSelection	8
Random	9
SpinHerding	4
SpinMoment	7
Function::WalshTransformTerm	2

6 Hierarchical Index

Chapter 3

Class Index

3.1 Class List

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

AdditiveGaussianNoise	
Additive Gaussian Noise	27
AffineMap	
Affine map	29
Algorithm	
Abstract search algorithm	31
BernoulliProcess	
Bernoulli process	34
BiasedCrossover	
Biased crossover	37
BinaryHerding	
·	38
BinaryMoment	
,	40
BmPbil	
	41
Cache	
	45
CallCounter	
	47
CompactGa	
, ,	48
CompleteSearch	
The second secon	50
Crossover	
	51
DeceptiveJump	
1 , 1	52
EqualProducts	_
	54
Error	_,
	56
ProgressTracker::Event	<u></u>
	57
Exception Basic exception	57

8 Class Index

Factorizat	ion	
	Factorization	58
	ntHillClimbing	
FourPeak	First ascent hill climbing	60
	s Four Peaks	61
Function		•
1	Function	64
FunctionC		
	Function controller	68
FunctionD	Function decorator	69
	MapComposition	00
	Composition of a function and a map	70
FunctionN		
	Function modifier	73
FunctionP	Plugin Function plugin	74
GeneticAl		74
	Genetic algorithm	75
Hamming		
	Hamming ball	78
Hamming		00
	Hamming sphere	80
	Hamming sphere neighborhood iterator	81
Hboa		
	Hierarchical Bayesian Optimization Algorithm	83
	ment, Herding >	
	Herding evolutionary algorithm	84
Hiff	Hierarchical if and only if	89
HncoEval	•	00
	Evaluator for HNCO functions	91
Hypercub		
	Hypercube iterator	92
IterativeAl		93
Iterator	Iterative search	90
	Iterator over bit vectors	95
Jump		
	Jump	97
Labs		00
LastEvalu	Low autocorrelation binary sequences	98
	Last evaluation	99
LeadingO	nes	
	3 · · ·	100
LinearFun		
LinearMar	Linear function	102
		104
LocalMax	·	
- 1	Local maximum	106
LongPath		
	Long path	107
Ltga	Linkage Tree Genetic Algorithm	108
	Limago noo deneno algonumi	100

3.1 Class List

Map
Map
MapComposition Map composition
MaximumReached
Maximum reached
MAX-SAT
Mmas Max-min ant system
Model
Model of a Boltzmann machine
Parameters of a Boltzmann machine
MuCommaLambdaEa (mu, lambda) EA
MultiBitFlip Multi bit flip
MuPlusLambdaEa
(mu+lambda) EA
Needle in a haystack
Negation Negation
Neighborhood
Neighborhood
Neighborhood iterator
NkLandscape NK landscape
NpsPbil
Population-based incremental learning with negative and positive selection
CallCounter with a limited number of evaluations
OneMax
OnePlusLambdaCommaLambdaGa (1+(lambda, lambda)) genetic algorithm
OnePlusOneEa
(1+1) EA
Parameter-less Population Pyramid
Pbil Population-based incremental learning
Permutation
Permutation
Plateau
PointValueException Point-value exception
Population Population
PriorNoise
Prior noise
ProgressTracker
PvAlgorithm Probability vector algorithm
•

10 Class Index

Qubo		
Dandan	Quadratic unconstrained binary optimization	167
Random	Random numbers	169
Randomi	LocalSearch	109
riandonn		170
Random	Search	
	Random search	172
Restart		
	Restart	173
Ridge	Ridge	171
Simulato	Ridge	174
Simulate		176
SingleBit		.,,
Ü		178
SingleBit	FlipIterator	
	5 1 5	179
SinusSu	mmationCancellation	
o:		181
SixPeaks		100
SpinHero		182
opini iero		184
SpinMon		
		187
Steepest	AscentHillClimbing	
	Steepest ascent hill climbing	188
StopOnM		
o. o -	· ·	189
StopOnT		100
Summati	Stop on target	192
Oumman		194
TargetRe		
J		197
Tournam	entSelection	
	•	198
Translation		
T	Translation	199
Trap	Trap	201
Umda	пар	201
Omaa	Univariate marginal distribution algorithm	203
UniformC	Crossover	
	Uniform crossover	205
WalshEx	pansion	
	1	206
WalshEx	pansion1	
\A/als = T	1 3	208
vvaisn⊨x	pansion2 Walsh expansion of degree 2	210
Function	::WalshTransformTerm	∠10
. anotion	Walsh transform term	212

Chapter 4

Namespace Documentation

4.1 hnco Namespace Reference

top-level HNCO namespace

Namespaces

· algorithm

Algorithms.

exception

Exceptions.

• function

Functions to be maximized.

neighborhood

Neighborhoods for local search.

random

Pseudo random numbers.

Classes

· class AffineMap

Affine map.

· class Hypercubelterator

Hypercube iterator.

· class Iterator

Iterator over bit vectors.

class LinearMap

Linear map.

• class Map

Мар.

• class MapComposition

Map composition.

class Permutation

Permutation.

· class Translation

Translation.

Types and functions related to bit matrices

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

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

      Display bit matrix.
• bool bm_is_valid (const bit_matrix_t &M)
      Check whether a bit matrix is valid.

    size_t bm_num_rows (const bit_matrix_t &M)

      Number of rows.

    size_t bm_num_columns (const bit_matrix_t &M)

      Number of columns.

    bool bm_is_square (const bit_matrix_t &M)

      Check whether the matrix is a square matrix.

    bool bm_is_identity (const bit_matrix_t &M)

      Check whether the matrix is the identity matrix.

    bool bm_is_upper_triangular (const bit_matrix_t &M)

      Check whether the matrix is upper triangular.

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

      Resize a bit matrix.

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

      Resize a bit matrix and make it a square matrix.

    void bm_clear (bit_matrix_t &M)

      Clear bit matrix.

    void bm identity (bit matrix t &M)

      Set the matrix to the identity matrix.

    void bm_random (bit_matrix_t &M)

      Sample a random bit matrix.

    void bm swap rows (bit matrix t &M, std::size t i, std::size t j)

      Swap two rows.

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

      Add two rows.

    void bm_row_echelon_form (bit_matrix_t &A)

      Compute a row echelon form of a matrix.

    std::size_t bm_rank (const bit_matrix_t &A)

      Compute the rank of a matrix.

    bool bm_solve (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system.

    bool bm_solve_upper_triangular (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system in upper triangular form.

    bool bm_invert (bit_matrix_t &M, bit_matrix_t &N)

      Invert a bit matrix.

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

      Multiply a bit matrix and a bit vector.

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

      Transpose.
```

Types and functions related to bit

```
typedef char bit_t

Bit.
bit_t bit_flip (bit_t b)

Flip bit.
```

Types and functions related to bit vectors

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

    typedef std::pair< bit_vector_t, double > point_value_t

      Type to represent point value pairs.

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

      Display bit vector.

    bool bv_is_valid (const bit_vector_t &x)

      Check whether the bit vector is valid.

    bool bv_is_zero (const bit_vector_t &x)

      Check whether the bit vector is zero.

    int bv_hamming_weight (const bit_vector_t &x)

     Hamming weight.

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

      Hamming weight.

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

     Hamming distance between two bit vectors.

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

      Dot product.

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

     Dot product.

    void bv_clear (bit_vector_t &x)

      Clear bit vector.

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

      Flip a single bit.

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

      Flip many bits.

    void bv_random (bit_vector_t &x)

      Sample a random bit vector.

    void bv_random (bit_vector_t &x, int k)

      Sample a random bit vector with given Hamming weight.

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

      Add two bit vectors.

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

      Add two bit vectors.

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

      Convert a bit vector to a bool vector.

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

      Convert a bool vector to a bit vector.

    std::size_t bv_to_size_type (const bit_vector_t &x)

      Convert a bit vector to a size_t.

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

      Convert a size_t to a bit vector.
```

Types and functions related to permutations

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

• bool perm_is_valid (const permutation_t &permutation)

Check that a vector represents a permutation.

void perm_random (permutation_t &s)

Sample a random permutation.

Types and functions related to sparse bit matrices

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

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

Display sparse bit matrix.

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

Convert a bit matrix to a sparse bit matrix.

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

Multiply a sparse bit matrix and a bit vector.

Types and functions related to sparse bit vectors

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

void bv_flip (bit_vector_t &x, const sparse_bit_vector_t &sbv)
 Flip many bits.

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

Display sparse bit vector.

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

Convert a bit vector to a sparse bit vector.

4.1.1 Detailed Description

top-level HNCO namespace

4.1.2 Typedef Documentation

4.1.2.1 bit_t

typedef char bit_t

Bit.

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

Definition at line 50 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 ( bit\_matrix\_t \ \& \ \textit{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 146 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 133 of file bit-vector.cc.

4.1.3.11 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 Mmas

Max-min ant system.

• class MuCommaLambdaEa

(mu, lambda) EA.

· class MuPlusLambdaEa

(mu+lambda) EA.

• class NpsPbil

Population-based incremental learning with negative and positive selection.

• class OnePlusLambdaCommaLambdaGa

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

· class OnePlusOneEa

(1+1) EA.

class Pbil

Population-based incremental learning.

class Population

Population.

class PvAlgorithm

Probability vector algorithm.

· class RandomLocalSearch

Random local search.

class RandomSearch

Random search.

· class Restart

Restart.

· class SimulatedAnnealing

Simulated annealing.

· class SteepestAscentHillClimbing

Steepest ascent hill climbing.

· class TournamentSelection

Population with tournament selection.

· class Umda

Univariate marginal distribution algorithm.

class UniformCrossover

Uniform crossover.

Functions

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

Check for symmetric matrix.

template<class T >

bool matrix_has_diagonal (const std::vector< std::vector< T >> &A, T x)

Check for diagonal elements.

template<class T >

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

Check for element range.

template < class T >

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

Check for element range.

template < class T >

T square (T x)

Generic square function.

• double logistic (double x)

Logistic function (sigmoid)

Type and functions related to probability vectors

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

Probability vector type.

double pv_entropy (const pv_t &pv)

Entropy of a probability vector.

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

Sample a bit vector.

• void pv_uniform (pv_t &pv)

Probability vector of the uniform distribution.

void pv_init (pv_t &pv)

Initialize.

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

Accumulate a bit vector.

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

Accumulate a bit vector.

void pv_average (pv_t &pv, int count)

Average

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

Update a probability vector toward a bit vector.

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

Update a probability vector toward a probability vector.

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

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

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

Bound the components of a probability vector.

4.2.1 Detailed Description

Algorithms.

4.3 hnco::algorithm::bm_pbil Namespace Reference

Boltzmann machine PBIL.

Classes

• class BmPbil

Boltzmann machine PBIL.

class Model

Model of a Boltzmann machine.

• class ModelParameters

Parameters of a Boltzmann machine.

4.3.1 Detailed Description

Boltzmann machine PBIL.

4.4 hnco::algorithm::hea Namespace Reference

Herding evolutionary algorithm.

Classes

· class BinaryHerding

Herding with binary variables.

struct BinaryMoment

Moment for binary variables.

class Hea

Herding evolutionary algorithm.

class SpinHerding

Herding with spin variables.

• struct SpinMoment

Moment for spin variables.

4.4.1 Detailed Description

Herding evolutionary algorithm.

4.5 hnco::exception Namespace Reference

Exceptions.

Classes

· class Error

Error.

class Exception

Basic exception.

class LastEvaluation

Last evaluation.

· class LocalMaximum

Local maximum.

class MaximumReached

Maximum reached.

• class PointValueException

Point-value exception.

class TargetReached

target reached

4.5.1 Detailed Description

Exceptions.

4.6 hnco::function Namespace Reference

Functions to be maximized.

Classes

· class AdditiveGaussianNoise

Additive Gaussian Noise.

· class Cache

Cache.

· class CallCounter

Call counter.

class DeceptiveJump

Deceptive jump.

class EqualProducts

Equal products.

· class Factorization

Factorization.

class FourPeaks

Four Peaks.

class Function

Function.

• class FunctionController

Function controller.

• class FunctionDecorator

Function decorator.

• class FunctionMapComposition

Composition of a function and a map.

· class FunctionModifier

Function modifier.

· class FunctionPlugin

Function plugin.

class Hiff

Hierarchical if and only if.

· class Jump

Jump.

• class Labs

Low autocorrelation binary sequences.

class LeadingOnes

Leading ones.

class LinearFunction

Linear function.

· class LongPath

Long path.

· class MaxSat

MAX-SAT.

• class Needle

Needle in a haystack.

· class Negation

Negation.

class NkLandscape

NK landscape.

class OnBudgetFunction

CallCounter with a limited number of evaluations.

class OneMax

OneMax.

· class Plateau

Plateau.

class PriorNoise

Prior noise.

class ProgressTracker

ProgressTracker.

· class Qubo

Quadratic unconstrained binary optimization.

• class Ridge

Ridge.

· class SinusSummationCancellation

Summation cancellation with sinus.

class SixPeaks

Six Peaks.

• class StopOnMaximum

Stop on maximum.

class StopOnTarget

Stop on target.

• class SummationCancellation

Summation cancellation.

class Trap

Trap.

class WalshExpansion

Walsh expansion.

class WalshExpansion1

Walsh expansion of degree 1.

• class WalshExpansion2

Walsh expansion of degree 2.

Functions

std::ostream & operator<< (std::ostream &stream, const ProgressTracker::Event &event)
 Insert formatted output.

4.6.1 Detailed Description

Functions to be maximized.

4.7 hnco::neighborhood Namespace Reference

Neighborhoods for local search.

Classes

• class BernoulliProcess

Bernoulli process.

class HammingBall

Hamming ball.

• class HammingSphere

Hamming sphere.

· class HammingSphereIterator

Hamming sphere neighborhood iterator.

class MultiBitFlip

Multi bit flip.

· class Neighborhood

Neighborhood.

· class NeighborhoodIterator

Neighborhood iterator.

· class SingleBitFlip

One bit neighborhood.

• class SingleBitFlipIterator

Single bit flip neighborhood iterator.

4.7.1 Detailed Description

Neighborhoods for local search.

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

4.8 hnco::random Namespace Reference

Pseudo random numbers.

Classes

• struct Random

Random numbers.

4.8.1 Detailed Description

Pseudo random numbers.

Chapter 5

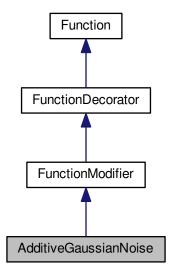
Class Documentation

5.1 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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

Inheritance diagram for AdditiveGaussianNoise:



Public Member Functions

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

Evaluate a bit vector.

Information about the function

```
• size_t get_bv_size ()

Get bit vector size.
```

• double get_maximum ()

Get the global maximum.

bool has_known_maximum ()

Check for a known maximum.

Private Attributes

 std::normal_distribution< double > _dist Normal distribution.

Additional Inherited Members

5.1.1 Detailed Description

Additive Gaussian Noise.

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

5.1.2 Member Function Documentation

```
5.1.2.1 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

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

```
5.1.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

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

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

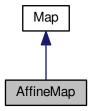
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

5.2 AffineMap Class Reference

Affine map.

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

Inheritance diagram for AffineMap:



Public Member Functions

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

Random instance.

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

Мар.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

Private Attributes

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

Friends

· class boost::serialization::access

5.2.1 Detailed Description

Affine map.

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

Definition at line 257 of file map.hh.

5.2.2 Member Function Documentation

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

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

int rows, int cols,

bool surjective)

Random instance.

Parameters

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

Exceptions



Definition at line 99 of file map.cc.

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

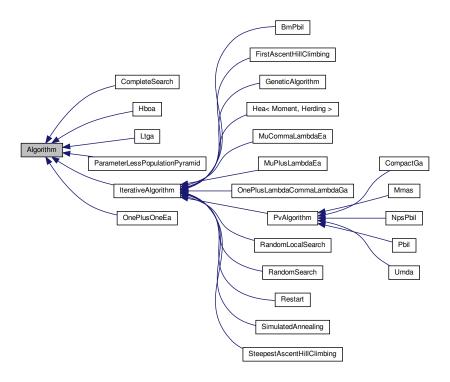
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.3 Algorithm Class Reference

Abstract search algorithm.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for Algorithm:



Public Member Functions

• Algorithm (int n)

Constructor.

virtual ∼Algorithm ()

Destructor.

Optimization

· virtual void init ()

Initialization.

• virtual void maximize ()=0

Maximize.

Getters

virtual const point_value_t & get_solution ()

Solution.

virtual size_t get_bv_size ()

Get bit vector size.

Setters

virtual void set_function (function::Function *function)

Set function.

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

Set functions.

void set_stream (std::ostream *x)

Output stream.

Protected Member Functions

• void random_solution ()

Random solution.

void set_solution (const bit_vector_t &x, double value)

Set solution.

void set_solution (const bit_vector_t &x)

Set solution.

• void update_solution (const bit_vector_t &x, double value)

Update solution (strict)

void update_solution (const point_value_t &pv)

Update solution (strict)

void update_solution (const bit_vector_t &x)

Update solution (strict).

Protected Attributes

• function::Function * function

Function.

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

Functions.

• point_value_t _solution

Solution.

Parameters

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

5.3.1 Detailed Description

Abstract search algorithm.

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

Definition at line 38 of file algorithm.hh.

5.3.2 Member Function Documentation

5.3.2.1 set_solution()

```
void set_solution ( {\tt const\ bit\_vector\_t\ \&\ x\ )} \quad [{\tt protected}]
```

Set solution.

Warning

Evaluates the function once.

Definition at line 47 of file algorithm.cc.

5.3.2.2 update_solution()

Update solution (strict).

Warning

Evaluates the function once.

Definition at line 70 of file algorithm.cc.

5.3.3 Member Data Documentation

5.3.3.1 _functions

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

Functions.

Each thread has its own function.

Definition at line 49 of file algorithm.hh.

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

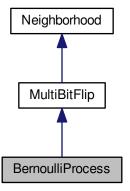
- lib/hnco/algorithms/algorithm.hh
- · lib/hnco/algorithms/algorithm.cc

5.4 BernoulliProcess Class Reference

Bernoulli process.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for BernoulliProcess:



Public Member Functions

• BernoulliProcess (int n)

Constructor.

• BernoulliProcess (int n, double p)

Constructor.

void set_probability (double p)

Set probability.

Private Member Functions

```
    void sample_bits ()
        Sample bits.
    void bernoulli_process ()
        Bernoulli process.
```

Private Attributes

```
    std::bernoulli_distribution _bernoulli_dist
        Bernoulli distribution (biased coin)
    std::binomial_distribution
    binomial_distribution.
    bool _reservoir_sampling = false
        Reservoir sampling.
```

Parameters

```
    bool _allow_stay = false
        Allow stay.
    void set_allow_stay (bool x)
        Set the flag _allow_stay.
```

Additional Inherited Members

5.4.1 Detailed Description

Bernoulli process.

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

Definition at line 220 of file neighborhood.hh.

5.4.2 Constructor & Destructor Documentation

Generated by Doxygen

Constructor.

Parameters

```
n Size of bit vectors
```

The Bernoulli probability is set to 1 / n.

Definition at line 255 of file neighborhood.hh.

5.4.2.2 BernoulliProcess() [2/2]

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

Constructor.

Parameters

n	Size of bit vectors
р	Bernoulli probability

Definition at line 265 of file neighborhood.hh.

5.4.3 Member Function Documentation

5.4.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

Definition at line 292 of file neighborhood.hh.

5.4.3.2 set_probability()

Set probability.

Sets _reservoir_sampling to true if E(X) < sqrt(n), where X is a random variable with a binomial distribution B(n, p), that is if np < sqrt(n) or p < 1 / sqrt(n).

Definition at line 276 of file neighborhood.hh.

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

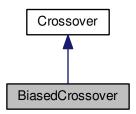
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.5 BiasedCrossover Class Reference

Biased crossover.

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

Inheritance diagram for BiasedCrossover:



Public Member Functions

• BiasedCrossover ()

Constructor.

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

Private Attributes

 std::bernoulli_distribution _bernoulli_dist Bernoulli distribution.

5.5.1 Detailed Description

Biased crossover.

Definition at line 75 of file crossover.hh.

5.5.2 Member Function Documentation

5.5.2.1 breed()

Breed.

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

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implements Crossover.

Definition at line 45 of file crossover.cc.

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

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

5.6 BinaryHerding Class Reference

Herding with binary variables.

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

Public Types

enum { DYNAMICS_MINIMIZE_NORM, DYNAMICS_MAXIMIZE_INNER_PRODUCT }

Public Member Functions

• BinaryHerding (int n)

Constructor.

• void init ()

Initialization.

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

Sample a bit vector.

• double error (const BinaryMoment &target)

Compute the error.

• double delta (const BinaryMoment &target)

Compute the norm of delta.

Setters

• void set_randomize_bit_order (bool x)

Randomize bit order.

void set_dynamics (int x)

Set the dynamics.

void set_weight (double x)

Set the weight of second order moments.

Protected Member Functions

void compute_delta (const BinaryMoment &target)

Compute delta.

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

Sample a bit vector.

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

Sample a bit vector.

Protected Attributes

· BinaryMoment _count

Counter moment.

• BinaryMoment _delta

Delta moment.

• permutation_t _permutation

Permutation.

• std::uniform_int_distribution< int > _choose_bit

Choose bit.

int _time

Time.

Parameters

• bool randomize bit order = false

Randomize bit order.

• int _dynamics = DYNAMICS_MINIMIZE_NORM

Dynamics.

• double _weight = 1

Weight of second order moments.

5.6.1 Detailed Description

Herding with binary variables.

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

5.6.2 Member Enumeration Documentation

5.6.2.1 anonymous enum

anonymous enum

Enumerator

DYNAMICS_MINIMIZE_NORM	Dynamics defined as minimization of a norm.
DYNAMICS_MAXIMIZE_INNER_PRODUCT	Dynamics defined as maximization of an inner product.
Generated by Doxygen	<u> </u>

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

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

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

5.7 BinaryMoment Struct Reference

Moment for binary variables.

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

Public Member Functions

• BinaryMoment (int n)

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

• void init ()

Initialize.

void add (const bit_vector_t &x)

Accumulate a bit vector.

· void average (int count)

Compute average.

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

Update moment.

• void bound (double margin)

Bound moment.

• double distance (const BinaryMoment &p) const

Distance.

• double norm_2 () const

Compute the norm 2.

· double diameter () const

Compute the diameter.

· size_t size () const

Size.

Public Attributes

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

Moment.

• double _weight = 1

Weight of second order moments.

5.8 BmPbil Class Reference 41

5.7.1 Detailed Description

Moment for binary variables.

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

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

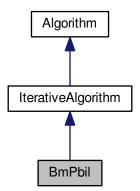
- · lib/hnco/algorithms/hea/moment-binary.hh
- lib/hnco/algorithms/hea/moment-binary.cc

5.8 BmPbil Class Reference

Boltzmann machine PBIL.

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

Inheritance diagram for BmPbil:



Public Types

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

Public Member Functions

• BmPbil (int n, int population_size)

Constructor.

· void init ()

Initialization.

Private Member Functions

· void iterate ()

Single iteration.

• void log ()

Log.

void sample (bit_vector_t &x)

Sample a bit vector.

• void sample_asynchronous ()

Asynchronous sampling.

• void sample_asynchronous_full_scan ()

Asynchronous sampling with full scan.

• void sample_synchronous ()

Synchronous sampling.

Private Attributes

• log_flags_t_log_flags

Log flags.

• Population _population

Population.

· Model _model

Model.

• ModelParameters _parameters_all

Parameters averaged over all individuals.

• ModelParameters_parameters_best

Parameters averaged over selected individuals.

ModelParameters _parameters_worst

Parameters averaged over negatively selected individuals.

• std::uniform_int_distribution< size_t > _choose_bit

Uniform distribution on bit_vector_t components.

· permutation_t _permutation

Permutation.

Parameters

• int _selection_size = 1

Selection size (number of selected individuals in the population)

• double _rate = 1e-3

Learning rate.

int _num_gs_steps = 100

Number of gibbs sampler steps.

• int _num_gs_cycles = 1

Number of gibbs sampler cycles.

• bool _negative_positive_selection = false

Negative and positive selection.

• int sampling = SAMPLING ASYNCHRONOUS

Sampling mode.

int _mc_reset_strategy = RESET_NO_RESET

MC reset strategy.

void set_selection_size (int x)

Set the selection size.

• void set_rate (double x)

Set the learning rate.

void set_num_gs_steps (int x)

Set the number of gibbs sampler steps.

• void set_num_gs_cycles (int x)

Set the number of gibbs sampler cycles.

• void set_negative_positive_selection (bool x)

Set negative and positive selection.

void set_sampling (int x)

Set the sampling mode.

void set_mc_reset_strategy (int x)

Set the MC reset strategy.

void set_log_flags (const log_flags_t &lf)

Set log flags.

Additional Inherited Members

5.8.1 Detailed Description

Boltzmann machine PBIL.

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

Reference:

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

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

5.8.2 Member Enumeration Documentation

5.8.2.1 anonymous enum

anonymous enum

Enumerator

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

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

5.8.2.2 anonymous enum

anonymous enum

Enumerator

SAMPLING_ASYNCHRONOUS	Asynchronous sampling. A single component of the internal state is randomly selected then updated by Gibbs sampling. This step is repeated _num_gs_steps times.
SAMPLING_ASYNCHRONOUS_FULL_SCAN	Asynchronous sampling with full scan. To sample a new bit vector, a random permutation is sampled and all components of the internal state are updated by Gibbs sampling in the order defined by the permutation.
SAMPLING_SYNCHRONOUS	Synchronous sampling. The full internal state is updated in one step from the probability vector made of the very marginal probabilities used in Gibbs sampling.

Definition at line 66 of file bm-pbil.hh.

5.8.2.3 anonymous enum

anonymous enum

Enumerator

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

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

5.8.3 Member Function Documentation

5.8.3.1 set_selection_size()

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

Set the selection size.

5.9 Cache Class Reference 45

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

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

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

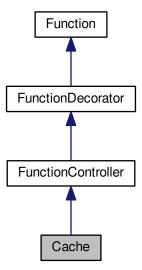
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

5.9 Cache Class Reference

Cache.

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

Inheritance diagram for Cache:



Public Member Functions

• Cache (Function *function)

Constructor.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

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

Private Attributes

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

Additional Inherited Members

5.9.1 Detailed Description

Cache.

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

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

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

5.9.2 Constructor & Destructor Documentation

```
5.9.2.1 Cache()

Cache (

Function * function ) [inline]

Constructor.

Parameters

function | Decorated function |
```

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

5.9.3 Member Function Documentation

5.9.3.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from FunctionController.

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

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

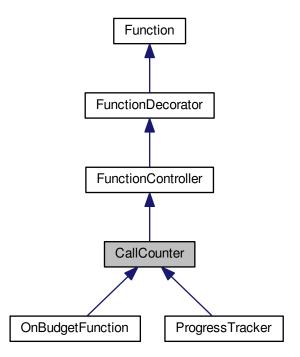
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.10 CallCounter Class Reference

Call counter.

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

Inheritance diagram for CallCounter:



Public Member Functions

• CallCounter (Function *function)

Constructor.

int get_num_calls ()

Get the number of calls.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Protected Attributes

int _num_calls

Number of calls.

5.10.1 Detailed Description

Call counter.

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

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

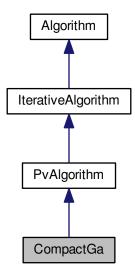
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.11 CompactGa Class Reference

Compact genetic algorithm.

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

Inheritance diagram for CompactGa:



Public Member Functions

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

Initialization.

Setters

void set_rate (double x)
 Set the learning rate.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

std::vector < bit_vector_t > _candidates
 Candidates.

Parameters

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

Additional Inherited Members

5.11.1 Detailed Description

Compact genetic algorithm.

Reference:

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

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

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

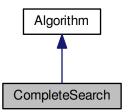
- · lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

5.12 CompleteSearch Class Reference

Complete search.

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

Inheritance diagram for CompleteSearch:



Public Member Functions

• CompleteSearch (int n)

Constructor.

• void maximize ()

Maximize.

Additional Inherited Members

5.12.1 Detailed Description

Complete search.

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

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

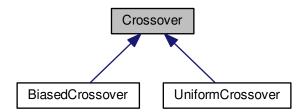
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

5.13 Crossover Class Reference

Crossover.

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

Inheritance diagram for Crossover:



Public Member Functions

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

5.13.1 Detailed Description

Crossover.

Definition at line 35 of file crossover.hh.

5.13.2 Member Function Documentation

5.13.2.1 breed()

Breed.

The offspring is the crossover of two parents.

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implemented in BiasedCrossover, and UniformCrossover.

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

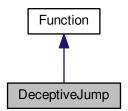
• lib/hnco/algorithms/ea/crossover.hh

5.14 DeceptiveJump Class Reference

Deceptive jump.

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

Inheritance diagram for DeceptiveJump:



Public Member Functions

• DeceptiveJump (int bv_size, int gap)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

```
• size_t _bv_size
```

Bit vector size.

int _gap

Gap.

5.14.1 Detailed Description

Deceptive jump.

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

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 84 of file jump.hh.

5.14.2 Member Function Documentation

```
5.14.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
_bv_size + _gap
```

Reimplemented from Function.

Definition at line 110 of file jump.hh.

5.14.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 106 of file jump.hh.

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

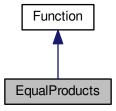
- lib/hnco/functions/jump.hh
- · lib/hnco/functions/jump.cc

5.15 EqualProducts Class Reference

Equal products.

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

Inheritance diagram for EqualProducts:



Public Member Functions

• EqualProducts ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• void random (int n, double upper_bound)

Random instance.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Member Functions

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

Private Attributes

std::vector< double > _numbers
 Numbers.

Friends

· class boost::serialization::access

5.15.1 Detailed Description

Equal products.

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

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

Reference:

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

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

5.15.2 Member Function Documentation

5.15.2.1 random()

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

Random instance.

Parameters

п	Size of bit vector
upper_bound	Upper bound of numbers

Generated by Doxygen

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

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

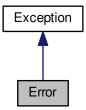
- · lib/hnco/functions/equal-products.hh
- · lib/hnco/functions/equal-products.cc

5.16 Error Class Reference

Error.

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

Inheritance diagram for Error:



Public Member Functions

• Error ()

Constructor.

• Error (const std::string &s)

Constructor.

virtual ∼Error ()

Destructor.

virtual const char * what () const

Get message.

Protected Attributes

std::string _what
 Message.

5.16.1 Detailed Description

Error.

Definition at line 83 of file exception.hh.

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

· lib/hnco/exception.hh

5.17 ProgressTracker::Event Struct Reference

Event.

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

Public Attributes

· int time

Time.

· double value

Value.

5.17.1 Detailed Description

Event.

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

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

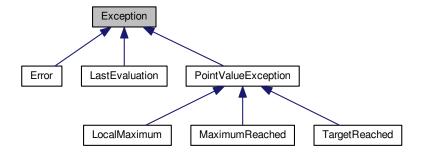
lib/hnco/functions/decorators/function-controller.hh

5.18 Exception Class Reference

Basic exception.

#include <hnco/exception.hh>

Inheritance diagram for Exception:



5.18.1 Detailed Description

Basic exception.

Definition at line 35 of file exception.hh.

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

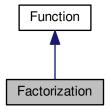
· lib/hnco/exception.hh

5.19 Factorization Class Reference

Factorization.

#include <hnco/functions/factorization.hh>

Inheritance diagram for Factorization:



Public Member Functions

• Factorization (std::string path)

Constructor.

∼Factorization ()

Destructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

· void display (std::ostream &stream)

Display.

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

Describe a bit vector.

Private Member Functions

void convert (const bit_vector_t &x)

Convert a bit vector into two numbers.

Private Attributes

```
• mpz_t _number
```

Number to factorize.

mpz_t _first_factor

First factor.

• mpz_t _second_factor

Second factor.

mpz_t _product

Product.

• std::string _first_factor_string

First factor in binary form.

std::string _second_factor_string

Secon factor in binary form.

• size_t _number_size

Number size in bits.

size_t _first_factor_size

First factor size in bits.

• size_t _second_factor_size

Second factor size in bits.

size_t _bv_size

Bit vector size.

5.19.1 Detailed Description

Factorization.

Reference:

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

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

Definition at line 28 of file factorization.hh.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 Factorization()

```
Factorization (
          std::string path )
```

Constructor.

Parameters

path Path to a file containing a number to factorize

Warning

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

Definition at line 16 of file factorization.cc.

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

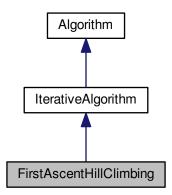
- · lib/hnco/functions/factorization.hh
- · lib/hnco/functions/factorization.cc

5.20 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

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

Inheritance diagram for FirstAscentHillClimbing:



Public Member Functions

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

Random initialization.

void init (const bit_vector_t &x)

Explicit initialization.

void init (const bit_vector_t &x, double value)

Explicit initialization.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

neighborhood::Neighborhoodlterator * _neighborhood.

Neighborhood.

5.20.1 Detailed Description

First ascent hill climbing.

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

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

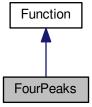
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc

5.21 FourPeaks Class Reference

Four Peaks.

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

Inheritance diagram for FourPeaks:



Public Member Functions

FourPeaks (int bv_size, int threshold)

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit vector t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

double get maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

· int _threshold

Threshold.

int maximum

Maximum.

5.21.1 Detailed Description

Four Peaks.

It is defined by

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

where:

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

This function has four maxima, of which exactly two are global ones.

For example, if n = 6 and t = 1:

- f(111111) = 6 (local maximum)
- f(111110) = 5
- f(111100) = 10 (global maximum)

Reference:

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

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

5.21.2 Member Function Documentation

5.21.2.1 get_maximum()

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

Get the global maximum.

Returns

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

Reimplemented from Function.

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

5.21.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

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

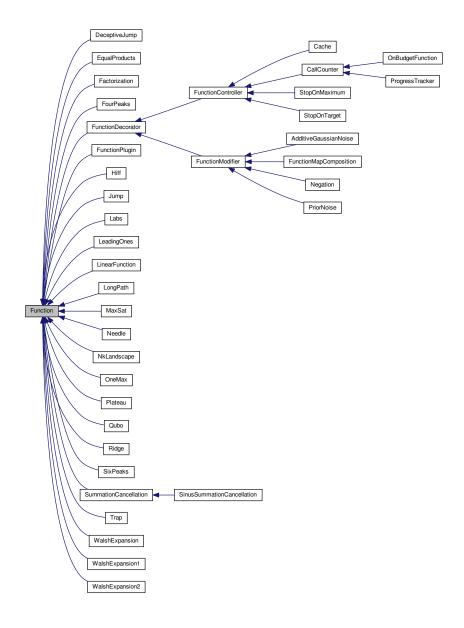
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

5.22 Function Class Reference

Function.

#include <hnco/functions/function.hh>

Inheritance diagram for Function:



Classes

struct WalshTransformTerm

Walsh transform term.

Public Member Functions

virtual ~Function ()
 Destructor.

Information about the function

virtual size_t get_bv_size ()=0

Get bit vector size.

virtual double get_maximum ()

Get the global maximum.

virtual bool has known maximum ()

Check for a known maximum.

· virtual bool provides incremental evaluation ()

Check whether the function provides incremental evaluation.

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

Compute the Walsh transform of the function.

Evaluation

virtual double eval (const bit_vector_t &)=0

Evaluate a bit vector.

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

Incremental evaluation.

virtual double safe_eval (const bit_vector_t &x)

Safely evaluate a bit vector.

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

Update after a safe evaluation.

Display

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

Display

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

Describe a bit vector.

5.22.1 Detailed Description

Function.

Definition at line 39 of file function.hh.

5.22.2 Member Function Documentation

5.22.2.1 compute_walsh_transform()

Compute the Walsh transform of the function.

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

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

Parameters

terms Vector of non zero terms of the Walsh transform

Warning

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

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

Definition at line 33 of file function.cc.

5.22.2.2 get_maximum()

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

Get the global maximum.

Exceptions

Error

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

Definition at line 78 of file function.hh.

5.22.2.3 incremental_eval()

Incremental evaluation.

Exceptions

Error

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

Definition at line 131 of file function.hh.

5.22.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

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

Definition at line 86 of file function.hh.

5.22.2.5 safe_eval()

Safely evaluate a bit vector.

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

Reimplemented in FunctionController.

Definition at line 141 of file function.hh.

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

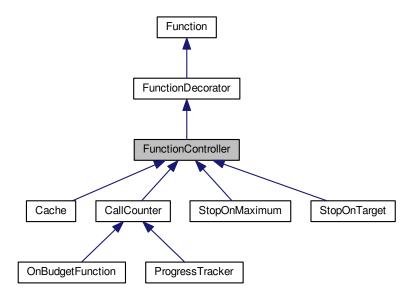
- · lib/hnco/functions/function.hh
- lib/hnco/functions/function.cc

5.23 FunctionController Class Reference

Function controller.

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

Inheritance diagram for FunctionController:



Public Member Functions

• FunctionController (Function *function)

Constructor.

Information about the function

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

Get the global maximum.

bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

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

Display

void display (std::ostream &stream)

Display

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

Describe a bit vector.

Additional Inherited Members

5.23.1 Detailed Description

Function controller.

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

5.23.2 Member Function Documentation

5.23.2.1 provides_incremental_evaluation()

bool provides_incremental_evaluation () [inline], [virtual]

Check whether the function provides incremental evaluation.

Returns

true if the decorated function does

Reimplemented from Function.

Reimplemented in Cache.

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

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

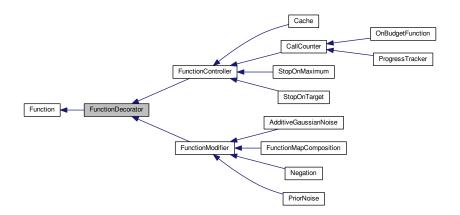
· lib/hnco/functions/decorators/function-controller.hh

5.24 FunctionDecorator Class Reference

Function decorator.

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

Inheritance diagram for FunctionDecorator:



Public Member Functions

• FunctionDecorator (Function *function)

Constructor.

Protected Attributes

• Function * _function

Decorated function.

5.24.1 Detailed Description

Function decorator.

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

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

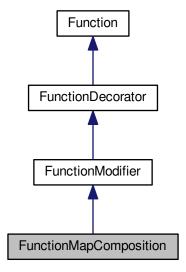
• lib/hnco/functions/decorators/function-decorator.hh

5.25 FunctionMapComposition Class Reference

Composition of a function and a map.

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

Inheritance diagram for FunctionMapComposition:



Public Member Functions

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

Constructor.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Information about the function

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

Private Attributes

Additional Inherited Members

5.25.1 Detailed Description

Composition of a function and a map.

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

5.25.2 Constructor & Destructor Documentation

5.25.2.1 FunctionMapComposition()

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

Constructor.

Precondition

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

_			
Exce	ntı	α n	10
	บแ	vi	ı

Error

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

5.25.3 Member Function Documentation

```
5.25.3.1 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

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

5.25.3.2 has_known_maximum()

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

Check for a known maximum.

Returns

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

Reimplemented from Function.

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

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

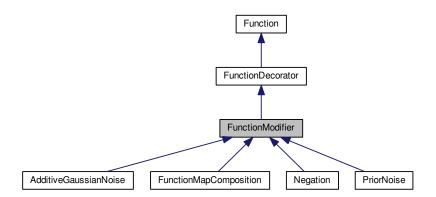
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

5.26 FunctionModifier Class Reference

Function modifier.

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

Inheritance diagram for FunctionModifier:



Public Member Functions

FunctionModifier (Function *function)
 Constructor.

Additional Inherited Members

5.26.1 Detailed Description

Function modifier.

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

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

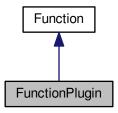
• lib/hnco/functions/decorators/function-modifier.hh

5.27 FunctionPlugin Class Reference

Function plugin.

#include <hnco/functions/plugin.hh>

Inheritance diagram for FunctionPlugin:



Public Member Functions

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

Constructor.

• ∼FunctionPlugin ()

Destructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Types

typedef double(* extern_function_t) (const char[], size_t)
 Type of an extern function.

Private Attributes

• size_t _bv_size

Bit vector size.

void * _handle

Handle returned by dlopen.

extern_function_t _extern_function

Extern function.

5.27.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 FunctionPlugin()

Constructor.

Parameters

bv_size	Size of bit vectors
path	Path to a shared library
name	Name of a function of the shared library

Definition at line 35 of file plugin.cc.

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

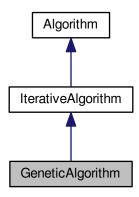
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

5.28 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



Public Member Functions

• GeneticAlgorithm (int n, int mu)

Constructor.

• void init ()

Initialization.

Setters

• void set_mutation_probability (double x)

Set the mutation probability.

• void set_crossover_probability (double x)

Set the crossover probability.

• void set_tournament_size (int x)

Set the tournament size.

void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

• TournamentSelection _parents

Parents.

• TournamentSelection _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

• std::bernoulli_distribution _do_crossover

Do crossover.

• UniformCrossover _crossover

Uniform crossover.

Parameters

```
    double _mutation_probability
    Mutation probability.
```

• double _crossover_probability = 0.5

Crossover probability.

• int _tournament_size = 10

Tournament size.

 bool _allow_stay = false Allow stay.

Additional Inherited Members

5.28.1 Detailed Description

Genetic algorithm.

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

Reference:

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

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

5.28.2 Constructor & Destructor Documentation

5.28.2.1 GeneticAlgorithm()

Constructor.

Parameters

n	Size of bit vectors
mu	Population size

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

5.28.3 Member Function Documentation

5.28.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

Definition at line 125 of file genetic-algorithm.hh.

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

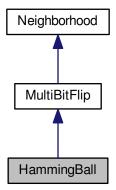
- · lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

5.29 HammingBall Class Reference

Hamming ball.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingBall:



Public Member Functions

HammingBall (int n, int r)
 Constructor.

Private Member Functions

• void sample_bits ()

Sample bits.

Private Attributes

• int _radius

Radius of the ball.

 $\bullet \quad \text{std::uniform_int_distribution} < \text{int} > _\text{choose_k} \\$

Choose the distance to the center.

Additional Inherited Members

5.29.1 Detailed Description

Hamming ball.

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

Definition at line 304 of file neighborhood.hh.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 HammingBall()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the ball

Definition at line 323 of file neighborhood.hh.

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

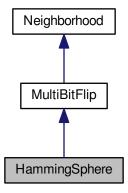
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.30 HammingSphere Class Reference

Hamming sphere.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingSphere:



Public Member Functions

• HammingSphere (int n, int r)

Constructor.

void set_radius (int r)

Set radius.

Private Member Functions

void sample_bits ()
 Sample bits.

Private Attributes

• int _radius

Radius of the sphere.

Additional Inherited Members

5.30.1 Detailed Description

Hamming sphere.

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

Definition at line 341 of file neighborhood.hh.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 HammingSphere()

```
\label{eq:hammingSphere} \begin{array}{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 357 of file neighborhood.hh.

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

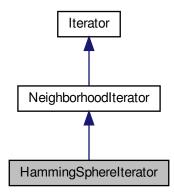
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.31 HammingSpherelterator Class Reference

Hamming sphere neighborhood iterator.

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

Inheritance diagram for HammingSphereIterator:



Public Member Functions

• HammingSphereIterator (int n, int r)

Constructor.

· bool has_next ()

Has next bit vector.

const bit_vector_t & next ()

Next bit vector.

Private Attributes

bit_vector_t _mask

Mutation mask.

· int radius

Radius of the ball.

· int _index

Index of the next bit to shift to the right.

· int weight

Partial Hamming weight.

Additional Inherited Members

5.31.1 Detailed Description

Hamming sphere neighborhood iterator.

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

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

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

5.32 Hboa Class Reference 83

5.31.2 Constructor & Destructor Documentation

5.31.2.1 HammingSphereIterator()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of Hamming Ball

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

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

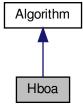
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.32 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

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

Inheritance diagram for Hboa:



Public Member Functions

• Hboa (int n)

Constructor.

• void maximize ()

Maximize.

void set_population_size (int n)

Set population size.

Private Attributes

• int _population_size = 10 Population size.

Additional Inherited Members

5.32.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 42 of file hboa.hh.

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

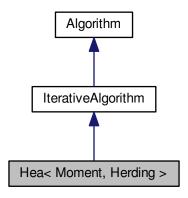
- · lib/hnco/algorithms/fast-efficient-p3/hboa.hh
- lib/hnco/algorithms/fast-efficient-p3/hboa.cc

5.33 Hea < Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



Public Types

- enum { RATE_CONSTANT, RATE_EXPONENTIAL, RATE_INVERSE }
- enum {

LOG_ERROR, LOG_DTU, LOG_DELTA, LOG_SELECTION,

LAST_LOG }

typedef std::bitset < LAST_LOG > log_flags_t

Type for log flags.

Public Member Functions

• Hea (int n, int population_size)

Constructor.

• void init ()

Initialization.

Setters

void set_herding (Herding *x)

Set the herding algorithm.

void set_margin (double x)

Set the moment margin.

void set_selection_size (int x)

Set the selection size.

void set_rate_strategy (int x)

Set the rate strategy.

void set_reset_period (int x)

Set the reset period.

void set_delay (int x)

Set the delay.

• void set_initial_rate (double x)

Set the initial value of the learning rate.

void set_time_constant (double x)

```
Set the time constant.
```

void set_bound_moment (bool x)

Set the bound moment after update.

void set_weight (double weight)

Set weight.

void set_log_flags (const log_flags_t &lf)

Set log flags.

Private Member Functions

· void iterate ()

Single iteration.

• void log ()

Log.

Private Attributes

Moment target

Moment.

• Moment selection

Moment of selected individuals.

Moment uniform

Uniform moment.

• algorithm::Population _population

Population.

Herding * _herding

Herding.

· double _error_cache

Error cache.

double _dtu_cache

Distance to uniform cache.

double _delta_cache

Delta cache.

· double selection cache

Selection distance cache.

log_flags_t _log_flags

Log flags.

Parameters

· double margin

Moment margin.

• int selection size = 1

Selection size.

int _rate_strategy = RATE_CONSTANT

Rate strategy.

• int reset period = 0

Reset period.

int _delay = 10000

Delay.

double _initial_rate = 1e-4

Initial value of the learning rate.

• double _time_constant = 1000

Time constant.

bool _bound_moment = false

Bound moment after update.

Additional Inherited Members

5.33.1 Detailed Description

Herding evolutionary algorithm.

Reference:

Arnaud Berny. 2015. Herding Evolutionary Algorithm. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15). ACM, New York, NY, USA, 1355–1356.

Definition at line 49 of file hea.hh.

5.33.2 Member Enumeration Documentation

5.33.2.1 anonymous enum

anonymous enum

Enumerator

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

Definition at line 54 of file hea.hh.

5.33.2.2 anonymous enum

anonymous enum

Enumerator

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

Definition at line 65 of file hea.hh.

5.33.3 Constructor & Destructor Documentation

Constructor.

Parameters

n Size of bit vectors

_margin is initialized to 1 / n.

Definition at line 234 of file hea.hh.

5.33.4 Member Function Documentation

```
5.33.4.1 set_reset_period()
```

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

Set the reset period.

Parameters

```
x Reset period
```

 $x \le 0$ means no reset.

Definition at line 281 of file hea.hh.

5.33.4.2 set_selection_size()

Set the selection size.

5.34 Hiff Class Reference 89

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

Definition at line 270 of file hea.hh.

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

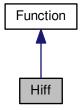
· lib/hnco/algorithms/hea/hea.hh

5.34 Hiff Class Reference

Hierarchical if and only if.

#include <hnco/functions/theory.hh>

Inheritance diagram for Hiff:



Public Member Functions

• Hiff (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

• size_t _depth

Tree depth.

5.34.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 165 of file theory.hh.

5.34.2 Member Function Documentation

```
5.34.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
(i + 1) * 2^i where 2^i = bv_size
```

Reimplemented from Function.

Definition at line 191 of file theory.hh.

5.34.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 187 of file theory.hh.

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

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

5.35 HncoEvaluator Class Reference

Evaluator for HNCO functions.

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

Inheritance diagram for HncoEvaluator:



Public Member Functions

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

Constructor.

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

Evaluate a bit vector.

Private Attributes

- hnco::function::Function * _function
 HNCO function.
- hnco::bit_vector_t _bv

Argument of HNCO function.

5.35.1 Detailed Description

Evaluator for HNCO functions.

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

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

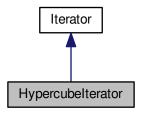
• lib/hnco/algorithms/fast-efficient-p3/hnco-evaluator.hh

5.36 Hypercubelterator Class Reference

Hypercube iterator.

#include <hnco/iterator.hh>

Inheritance diagram for Hypercubelterator:



Public Member Functions

• Hypercubelterator (int n)

Constructor.

bool has_next ()

Has next bit vector.

• const bit_vector_t & next ()

Next bit vector.

Additional Inherited Members

5.36.1 Detailed Description

Hypercube iterator.

Implemented as a simple binary adder.

Definition at line 69 of file iterator.hh.

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

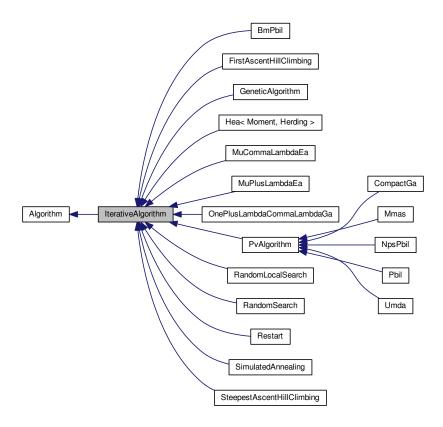
- · lib/hnco/iterator.hh
- lib/hnco/iterator.cc

5.37 Iterative Algorithm Class Reference

Iterative search.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for IterativeAlgorithm:



Public Member Functions

• IterativeAlgorithm (int n)

Constructor.

• void maximize ()

Maximize.

Setters

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

Protected Member Functions

virtual void iterate ()=0
 Single iteration.

• virtual void log ()

Log

Protected Attributes

```
· int _iteration
```

Current iteration.

• bool _something_to_log Something to log.

Parameters

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

5.37.1 Detailed Description

Iterative search.

Definition at line 145 of file algorithm.hh.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 IterativeAlgorithm()

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 175 of file algorithm.hh.

5.37.3 Member Function Documentation

5.37.3.1 maximize()

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

Maximize.

Inside the loop:

- call iterate()
- call log()

Warning

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

Implements Algorithm.

Definition at line 77 of file algorithm.cc.

5.37.3.2 set_num_iterations()

Set the number of iterations.

Parameters

```
x Number of iterations
```

 $x \le 0$ means indefinite

Definition at line 199 of file algorithm.hh.

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

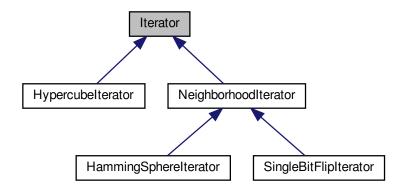
- · lib/hnco/algorithms/algorithm.hh
- · lib/hnco/algorithms/algorithm.cc

5.38 Iterator Class Reference

Iterator over bit vectors.

#include <hnco/iterator.hh>

Inheritance diagram for Iterator:



Public Member Functions

• Iterator (int n)

Constructor.

virtual ∼lterator ()

Destructor.

· virtual void init ()

Initialization.

• virtual bool has_next ()=0

Has next bit vector.

• virtual const bit_vector_t & next ()=0

Next bit vector.

Protected Attributes

bit_vector_t _current

Current bit vector.

• bool <u>_initial_state</u> = true

Flag for initial state.

5.38.1 Detailed Description

Iterator over bit vectors.

Definition at line 34 of file iterator.hh.

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

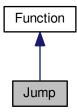
· lib/hnco/iterator.hh

5.39 Jump Class Reference

Jump.

#include <hnco/functions/jump.hh>

Inheritance diagram for Jump:



Public Member Functions

• Jump (int bv_size, int gap)

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

· size_t _bv_size

Bit vector size.

int _gap

Gap.

5.39.1 Detailed Description

Jump.

Reference:

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

Definition at line 40 of file jump.hh.

5.39.2 Member Function Documentation

```
5.39.2.1 get_maximum()

double get_maximum ( ) [inline], [virtual]

Get the global maximum.

Returns
   _bv_size
```

Reimplemented from Function.

Definition at line 66 of file jump.hh.

```
5.39.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 62 of file jump.hh.

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

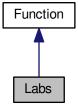
- · lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

5.40 Labs Class Reference

Low autocorrelation binary sequences.

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

Inheritance diagram for Labs:



Public Member Functions

• Labs (int n)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Attributes

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

5.40.1 Detailed Description

Low autocorrelation binary sequences.

Reference:

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

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

Definition at line 43 of file labs.hh.

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

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

5.41 LastEvaluation Class Reference

Last evaluation.

#include <hnco/exception.hh>

Inheritance diagram for LastEvaluation:



5.41.1 Detailed Description

Last evaluation.

Definition at line 79 of file exception.hh.

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

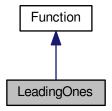
· lib/hnco/exception.hh

5.42 LeadingOnes Class Reference

Leading ones.

#include <hnco/functions/theory.hh>

Inheritance diagram for LeadingOnes:



Public Member Functions

• LeadingOnes (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

5.42.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 93 of file theory.hh.

5.42.2 Member Function Documentation

```
5.42.2.1 get_maximum()
```

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 117 of file theory.hh.

5.42.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 113 of file theory.hh.

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

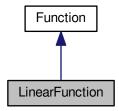
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.43 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



Public Member Functions

• LinearFunction ()

Constructor.

• void random (int n)

Random instance.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Member Functions

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

Private Attributes

std::vector< double > _weights
 Weights.

Friends

· class boost::serialization::access

5.43.1 Detailed Description

Linear function.

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

5.43.2 Member Function Documentation

5.43.2.1 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 76 of file linear-function.hh.

5.43.2.2 random()

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

Random instance.

Parameters

```
n Size of bit vectors
```

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

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

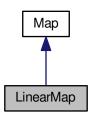
- · lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

5.44 LinearMap Class Reference

Linear map.

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

Inheritance diagram for LinearMap:



Public Member Functions

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

Random instance.

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

Мар.

size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

• bit_matrix_t _bm

Bit matrix.

Friends

· class boost::serialization::access

5.44.1 Detailed Description

Linear map.

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

Definition at line 193 of file map.hh.

5.44.2 Member Function Documentation

5.44.2.1 is_surjective()

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

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 90 of file map.cc.

5.44.2.2 random()

Random instance.

Parameters

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

Exceptions

Error

Definition at line 61 of file map.cc.

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

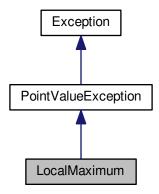
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.45 LocalMaximum Class Reference

Local maximum.

#include <hnco/exception.hh>

Inheritance diagram for LocalMaximum:



Public Member Functions

LocalMaximum (const point_value_t &pv)
 Const.

Additional Inherited Members

5.45.1 Detailed Description

Local maximum.

Definition at line 70 of file exception.hh.

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

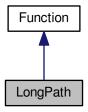
• lib/hnco/exception.hh

5.46 LongPath Class Reference

Long path.

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

Inheritance diagram for LongPath:



Public Member Functions

• LongPath (int bv_size, int prefix_length)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Attributes

size_t _bv_size

Bit vector size.

· int _prefix_length

Prefix length.

5.46.1 Detailed Description

Long path.

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

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

- 0000
- 0001

- 0011
- 0111
- 1111
- 1101
- 1100

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

References:

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

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

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

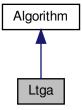
- · lib/hnco/functions/long-path.hh
- · lib/hnco/functions/long-path.cc

5.47 Ltga Class Reference

Linkage Tree Genetic Algorithm.

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

Inheritance diagram for Ltga:



Public Member Functions

• Ltga (int n)

Constructor.

• void maximize ()

Maximize.

void set_population_size (int n)

Set population size.

Private Attributes

• int _population_size = 10 Population size.

Additional Inherited Members

5.47.1 Detailed Description

Linkage Tree Genetic Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 40 of file Itga.hh.

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

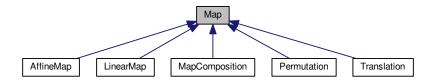
- · lib/hnco/algorithms/fast-efficient-p3/ltga.hh
- lib/hnco/algorithms/fast-efficient-p3/ltga.cc

5.48 Map Class Reference

Map.

#include <hnco/map.hh>

Inheritance diagram for Map:



Public Member Functions

virtual ∼Map ()

Destructor.

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

Map

virtual size_t get_input_size ()=0

Get input size.

• virtual size_t get_output_size ()=0

Get output size.

• virtual bool is_surjective ()

Check for surjective map.

5.48.1 Detailed Description

Мар.

Definition at line 39 of file map.hh.

5.48.2 Member Function Documentation

```
5.48.2.1 is_surjective()
```

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

Check for surjective map.

Returns

false

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

Definition at line 59 of file map.hh.

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

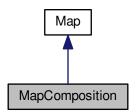
· lib/hnco/map.hh

5.49 MapComposition Class Reference

Map composition.

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

Inheritance diagram for MapComposition:



Public Member Functions

```
• MapComposition ()
```

Default constructor.

MapComposition (Map *outer, Map *inner)

Constructor.

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

Man.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Attributes

```
Map * _outer
```

Outer map.

• Map * _inner

Inner map.

bit_vector_t _bv

Temporary bit vector.

5.49.1 Detailed Description

Map composition.

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

Definition at line 327 of file map.hh.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 MapComposition()

Constructor.

Parameters

outer	outer map
inner	inner map

Precondition

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

Definition at line 351 of file map.hh.

5.49.3 Member Function Documentation

5.49.3.1 is_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 375 of file map.hh.

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

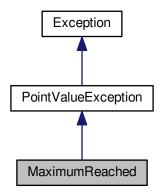
· lib/hnco/map.hh

5.50 MaximumReached Class Reference

Maximum reached.

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

Inheritance diagram for MaximumReached:



Public Member Functions

MaximumReached (const point_value_t &pv)
 Constructor.

Additional Inherited Members

5.50.1 Detailed Description

Maximum reached.

Definition at line 52 of file exception.hh.

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

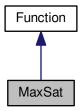
• lib/hnco/exception.hh

5.51 MaxSat Class Reference

MAX-SAT.

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

Inheritance diagram for MaxSat:



Public Member Functions

MaxSat ()

Default constructor.

• void random (int n, int k, int c)

Random instance.

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

Random instance with satisfiable expression.

void load (std::istream &stream)

Load an instance.

void save (std::ostream &stream)

Save an instance.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

· void display (std::ostream &stream)

Display the expression.

Private Attributes

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

• size_t _num_variables

Number of variables.

5.51.1 Detailed Description

MAX-SAT.

Reference:

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

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

5.51.2 Member Function Documentation

```
5.51.2.1 load()
```

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

Load an instance.

Exceptions

Error

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

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

```
void random (
              int n_{,}
              int k,
              int c)
```

Random instance.

Parameters

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

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

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

Random instance with satisfiable expression.

Warning

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

Parameters

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

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

5.51.3 Member Data Documentation

```
5.51.3.1 _expression

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

Expression.

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

Definition at line 52 of file max-sat.hh.

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

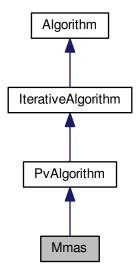
- · lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.52 Mmas Class Reference

Max-min ant system.

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

Inheritance diagram for Mmas:



Public Member Functions

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

Initialization.

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

5.53 Model Class Reference 117

Protected Attributes

bit_vector_t _x
 Candidate solution.

Parameters

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

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

Additional Inherited Members

5.52.1 Detailed Description

Max-min ant system.

Reference:

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

Definition at line 41 of file mmas.hh.

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

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

5.53 Model Class Reference

Model of a Boltzmann machine.

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

Public Member Functions

```
• Model (int n)
```

Constructor.

void init ()

Initialize.

· void reset_mc ()

Reset Markov chain.

void gibbs_sampler (size_t i)

A Gibbs sampler cycle.

void gibbs_sampler_synchronous ()

A synchronous Gibbs sampler.

const bit_vector_t & get_state ()

Get the state of the Gibbs sampler.

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

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

• double norm_infinite ()

Infinite norm of the parameters.

• double norm_l1 ()

I1 norm of the parameters

Private Attributes

• ModelParameters _model_parameters

Model parameters.

bit_vector_t _state

State of the Gibbs sampler.

pv_t _pv

Probability vector for synchronous Gibbs sampling.

5.53.1 Detailed Description

Model of a Boltzmann machine.

Definition at line 75 of file model.hh.

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

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

5.54 ModelParameters Class Reference

Parameters of a Boltzmann machine.

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

Public Member Functions

• ModelParameters (int n)

Constructor.

void init ()

Initialize.

void add (const bit_vector_t &x)

Add a bit_vector_t.

· void average (int count)

Compute averages.

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

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

• double norm infinite ()

Infinite norm of the parameters.

• double norm_I1 ()

I1 norm of the parameters

Private Attributes

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

std::vector< double > _bias

Bias.

Weights.

Friends

· class Model

5.54.1 Detailed Description

Parameters of a Boltzmann machine.

Definition at line 36 of file model.hh.

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

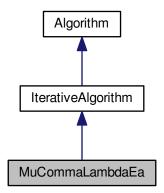
- · lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

5.55 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



Public Member Functions

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

Constructor.

· void init ()

Initialization.

Setters

• void set_mutation_probability (double x)

Set the mutation probability.

void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

• void iterate ()

Single iteration.

Private Attributes

· Population _parents

Parents.

• Population _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

std::uniform_int_distribution< int > _select_parent

Select parent.

Parameters

• double _mutation_probability

Mutation probability.

• bool <u>_allow_stay</u> = false

Allow stay.

Additional Inherited Members

5.55.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

5.55.2 Constructor & Destructor Documentation

5.55.2.1 MuCommaLambdaEa()

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

Constructor.

Parameters

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

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

5.55.3 Member Function Documentation

5.55.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

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

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

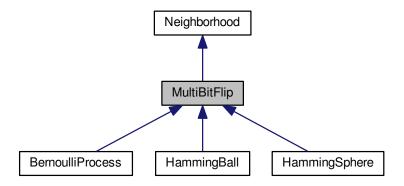
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- · lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

5.56 MultiBitFlip Class Reference

Multi bit flip.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for MultiBitFlip:



Public Member Functions

• MultiBitFlip (int n)

Constructor.

Protected Member Functions

void bernoulli_trials (int k)

Sample a given number of bits using Bernoulli trials.

void reservoir_sampling (int k)

Sample a given number of bits using resevoir sampling.

Additional Inherited Members

5.56.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 MultiBitFlip()

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

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 206 of file neighborhood.hh.

5.56.3 Member Function Documentation

5.56.3.1 bernoulli_trials()

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

Sample a given number of bits using Bernoulli trials.

Parameters

k Number of bits to sample

Definition at line 34 of file neighborhood.cc.

5.56.3.2 reservoir_sampling()

Sample a given number of bits using resevoir sampling.

Parameters

k Number of bits to sample

Definition at line 52 of file neighborhood.cc.

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

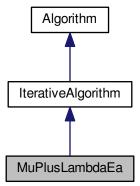
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.57 MuPlusLambdaEa Class Reference

(mu+lambda) EA.

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

Inheritance diagram for MuPlusLambdaEa:



Public Member Functions

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

Constructor.

· void init ()

Initialization.

Setters

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

```
Set the mutation probability.
```

void set_allow_stay (bool x)

Set the flag _allow_stay.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

· Population _parents

Parents.

• Population _offsprings

Offsprings.

• neighborhood::BernoulliProcess _mutation

Mutation operator.

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

Select parent.

Parameters

• double _mutation_probability

Mutation probability.

• bool <u>_allow_stay</u> = false

Allow stay.

Additional Inherited Members

5.57.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

5.57.2 Constructor & Destructor Documentation

5.57.2.1 MuPlusLambdaEa()

Constructor.

Parameters

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

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

5.57.3 Member Function Documentation

5.57.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

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

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

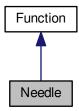
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

5.58 Needle Class Reference

Needle in a haystack.

#include <hnco/functions/theory.hh>

Inheritance diagram for Needle:



Public Member Functions

Needle (int bv_size)

 ${\it Constructor.}$

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

• size_t _bv_size

Bit vector size.

5.58.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 129 of file theory.hh.

5.58.2 Member Function Documentation

```
5.58.2.1 get_maximum()
double get_maximum ( ) [inline], [virtual]
Get the global maximum.
Returns
     1
Reimplemented from Function.
Definition at line 153 of file theory.hh.
5.58.2.2 has_known_maximum()
bool has_known_maximum ( ) [inline], [virtual]
Check for a known maximum.
Returns
     true
Reimplemented from Function.
Definition at line 149 of file theory.hh.
```

• lib/hnco/functions/theory.hh

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

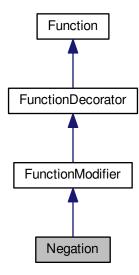
• lib/hnco/functions/theory.cc

5.59 Negation Class Reference

Negation.

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

Inheritance diagram for Negation:



Public Member Functions

• Negation (Function *function)

Constructor.

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

Additional Inherited Members

5.59.1 Detailed Description

Negation.

Use cases:

- for algorithms which minimize rather than maximize a function
- · for functions one wishes to minimize
- · when minimization is needed inside an algorithm

Definition at line 58 of file function-modifier.hh.

5.59.2 Member Function Documentation

```
5.59.2.1 get_maximum()
```

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

Get the global maximum.

Exceptions

Error

Reimplemented from Function.

Definition at line 76 of file function-modifier.hh.

```
5.59.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

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

5.59.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

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

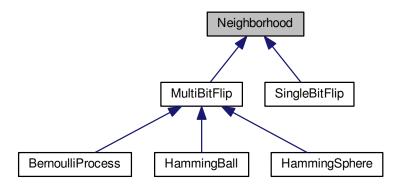
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

5.60 Neighborhood Class Reference

Neighborhood.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for Neighborhood:



Public Member Functions

```
    Neighborhood (int n)
```

Constructor.

• virtual ∼Neighborhood ()

Destructor.

virtual void set_origin (const bit_vector_t &x)

Set the origin.

virtual const bit_vector_t & get_origin ()

Get the origin.

virtual const bit_vector_t & get_candidate ()

Get the candidate bit vector.

virtual const sparse_bit_vector_t & get_flipped_bits ()

Get flipped bits.

• virtual void propose ()

Propose a candidate bit vector.

· virtual void keep ()

Keep the candidate bit vector.

· virtual void forget ()

Forget the candidate bit vector.

virtual void mutate (bit_vector_t &bv)

Mutate.

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

Мар.

Protected Member Functions

• virtual void sample_bits ()=0

Sample bits.

Protected Attributes

· bit_vector_t _origin

Origin of the neighborhood.

• bit_vector_t _candidate

candidate bit vector

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

Uniform index distribution.

• sparse_bit_vector_t _flipped_bits

Flipped bits.

5.60.1 Detailed Description

Neighborhood.

A neighborhood maintains two points, _origin and _candidate. They are initialized in the same state by set_origin. A Neighborhood class must implement the member function sample_bits which samples the bits to flip in _origin to get a _candidate. The following member functions take care of the modifications:

```
· propose: flip _candidate
```

- · keep: flip _origin
- · forget flip _candidate

After keep or forget, _origin and _candidate are in the same state again.

A Neighborhood class can also behave as a mutation operator through the member functions mutate and map.

Definition at line 61 of file neighborhood.hh.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 Neighborhood()

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

Constructor.

Parameters

```
n Size of bit vectors
```

Definition at line 86 of file neighborhood.hh.

5.60.3 Member Function Documentation

Мар.

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

Parameters

input	Input bit vector
output	Output bit vector

Definition at line 148 of file neighborhood.hh.

5.60.3.2 mutate()

Mutate.

In-place mutation of the bit vector.

Parameters

Definition at line 134 of file neighborhood.hh.

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

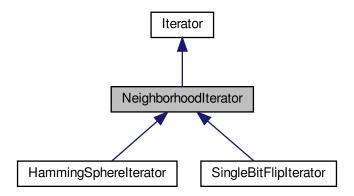
• lib/hnco/neighborhoods/neighborhood.hh

5.61 NeighborhoodIterator Class Reference

Neighborhood iterator.

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

Inheritance diagram for NeighborhoodIterator:



Public Member Functions

NeighborhoodIterator (int n)

Constructor.

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

Additional Inherited Members

5.61.1 Detailed Description

Neighborhood iterator.

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

5.61.2 Constructor & Destructor Documentation

5.61.2.1 NeighborhoodIterator()

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

Constructor.

Parameters

```
n Size of bit vectors
```

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

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

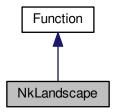
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.62 NkLandscape Class Reference

NK landscape.

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

Inheritance diagram for NkLandscape:



Public Member Functions

• NkLandscape ()

Default constructor.

• size_t get_bv_size ()

Get bit vector size.

void random (int n, int k, double stddev)

Random instance.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Member Functions

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

Private Attributes

- std::vector < std::vector < int > > _neighbors
 Bit neighbors.
- std::vector< std::vector< double > __partial_functions
 Partial functions.

Friends

· class boost::serialization::access

5.62.1 Detailed Description

NK landscape.

Reference:

S. A. Kauffman. 1993. The origins of order: self-organisation and selection in evolution. Oxford University Press. Definition at line 47 of file nk-landscape.hh.

5.62.2 Member Function Documentation

5.62.2.1 random()

Random instance.

Parameters

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

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

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

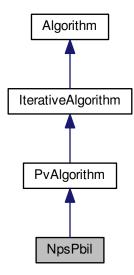
- lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

5.63 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

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

Inheritance diagram for NpsPbil:



Public Member Functions

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

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

- pv_t _mean_best

Mean of best individuals.

pv_t _mean_worst

Mean of worst individuals.

Parameters

• int _selection_size = 1

Selection size.

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

Additional Inherited Members

5.63.1 Detailed Description

Population-based incremental learning with negative and positive selection.

Reference:

Arnaud Berny. 2001. Extending selection learning toward fixed-length d-ary strings. In Artificial Evolution (Lecture Notes in Computer Science), P. Collet and others (Eds.). Springer, Le Creusot.

Definition at line 41 of file nps-pbil.hh.

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

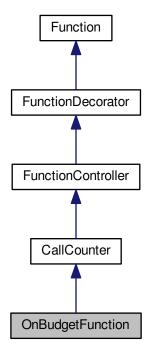
- lib/hnco/algorithms/pv/nps-pbil.hh
- lib/hnco/algorithms/pv/nps-pbil.cc

5.64 OnBudgetFunction Class Reference

CallCounter with a limited number of evaluations.

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

Inheritance diagram for OnBudgetFunction:



Public Member Functions

• OnBudgetFunction (Function *function, int budget)

Constructor.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Private Attributes

· int _budget

Budget.

Additional Inherited Members

5.64.1 Detailed Description

CallCounter with a limited number of evaluations.

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

5.64.2 Member Function Documentation

Evaluate a bit vector.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.64.2.2 incremental_eval()

Incremental evaluation.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

5.64.2.3 update()

Update after a safe evaluation.

Exceptions

LastEvaluation

Reimplemented from CallCounter.

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

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

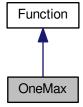
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.65 OneMax Class Reference

OneMax.

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

Inheritance diagram for OneMax:



Public Member Functions

OneMax (int bv_size)
 Constructor.

Information about the function

```
• size_t get_bv_size ()
```

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

• double incremental_eval (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits)

Incremental evaluation.

Private Attributes

• size_t _bv_size

Bit vector size.

5.65.1 Detailed Description

OneMax.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 36 of file theory.hh.

5.65.2 Member Function Documentation

```
5.65.2.1 get_maximum()
```

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

Get the global maximum.

Returns

bv size

Reimplemented from Function.

Definition at line 57 of file theory.hh.

5.65.2.2 has_known_maximum()

bool has_known_maximum () [inline], [virtual]

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 61 of file theory.hh.

5.65.2.3 provides_incremental_evaluation()

bool provides_incremental_evaluation () [inline], [virtual]

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 66 of file theory.hh.

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

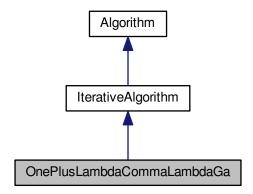
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.66 OnePlusLambdaCommaLambdaGa Class Reference

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

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

Inheritance diagram for OnePlusLambdaCommaLambdaGa:



Public Member Functions

• OnePlusLambdaCommaLambdaGa (int n, int lambda)

Constructor.

• void init ()

Initialization.

Setters

• void set_mutation_probability (double x)

Set the mutation probability.

void set_crossover_bias (double x)

Set the crossover bias.

Private Member Functions

· void iterate ()

Single iteration.

Private Attributes

· Population _offsprings

Offsprings.

• std::binomial_distribution< int > _radius_dist

Radius distribution.

· neighborhood::HammingSphere _mutation

Mutation operator.

· bit_vector_t _parent

Parent.

• BiasedCrossover _crossover

Biased crossover.

Parameters

- double _mutation_probability
 Mutation probability.
- double _crossover_bias

Crossover bias.

Additional Inherited Members

5.66.1 Detailed Description

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

Reference:

Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. Theoretical Computer Science 567 (2015), 87–104.

Definition at line 49 of file one-plus-lambda-comma-lambda-ga.hh.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 OnePlusLambdaCommaLambdaGa()

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

Constructor.

By default, _mutation_probability is set to lambda / n and _crossover_bias to 1 / lambda.

Parameters

n	Size of bit vectors
lambda	Offspring population size

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

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

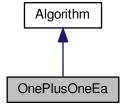
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

5.67 OnePlusOneEa Class Reference

(1+1) EA.

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

Inheritance diagram for OnePlusOneEa:



Public Member Functions

• OnePlusOneEa (int n)

Constructor.

void set_function (function::Function *function)

Set function.

· void init ()

Initialization.

· void maximize ()

Maximize.

Setters

• void set_num_iterations (int x)

Set the number of iterations.

void set_mutation_probability (double x)

Set the mutation probability.

void set_allow_stay (bool x)

Set the flag _allow_stay.

• void set_incremental_evaluation (bool x)

Set incremental evaluation.

Private Attributes

• neighborhood::BernoulliProcess _neighborhood

Neighborhood.

RandomLocalSearch _rls

Random local search.

Parameters

• int num iterations = 0

Number of iterations.

· double _mutation_probability

Mutation probability.

bool <u>_allow_stay</u> = false

Allow stay.

• bool _incremental_evaluation = false

Incremental evaluation.

Additional Inherited Members

5.67.1 Detailed Description

(1+1) EA.

(1+1) EA is implemented as a RandomLocalSearch with a BernoulliProcess neighborhood and infinite patience. Thus it does derive from IterativeAlgorithm.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

5.67.2 Constructor & Destructor Documentation

5.67.2.1 OnePlusOneEa()

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

Constructor.

Parameters

```
n Size of bit vectors
```

_mutation_probability is initialized to 1 / n.

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

5.67.3 Member Function Documentation

5.67.3.1 set_allow_stay()

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

Set the flag _allow_stay.

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

Definition at line 126 of file one-plus-one-ea.hh.

5.67.3.2 set_num_iterations()

Set the number of iterations.

Parameters

x Number of iterations

x <= 0 means indefinite

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

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

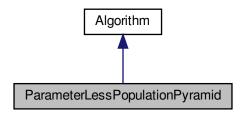
· lib/hnco/algorithms/ea/one-plus-one-ea.hh

5.68 ParameterLessPopulationPyramid Class Reference

Parameter-less Population Pyramid.

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

Inheritance diagram for ParameterLessPopulationPyramid:



Public Member Functions

· ParameterLessPopulationPyramid (int n)

Constructor.

· void maximize ()

Maximize.

Additional Inherited Members

5.68.1 Detailed Description

Parameter-less Population Pyramid.

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

Author: Brian W. Goldman

Reference:

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

Integrated into HNCO by Arnaud Berny

Definition at line 44 of file p3.hh.

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

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

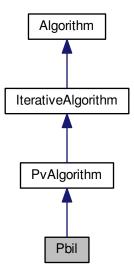
5.69 Pbil Class Reference 149

5.69 Pbil Class Reference

Population-based incremental learning.

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

Inheritance diagram for Pbil:



Public Member Functions

- Pbil (int n, int population_size)

 Constructor.
- void init ()

Initialization.

Setters

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

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

• Population _population

Population.

pv_t _mean

Mean of selected bit vectors.

Parameters

- int _selection_size = 1 Selection size.
- double <u>_rate</u> = 1e-3 <u>Learning rate</u>.

Additional Inherited Members

5.69.1 Detailed Description

Population-based incremental learning.

Reference:

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

Definition at line 40 of file pbil.hh.

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

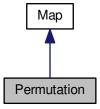
- · lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

5.70 Permutation Class Reference

Permutation.

#include <hnco/map.hh>

Inheritance diagram for Permutation:



Public Member Functions

```
• void random (int n)
```

Random instance.

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

Мар.

• size_t get_input_size ()

Get input size.

• size_t get_output_size ()

Get output size.

• bool is_surjective ()

Check for surjective map.

Private Member Functions

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

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

Private Attributes

Friends

· class boost::serialization::access

5.70.1 Detailed Description

Permutation.

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

Definition at line 132 of file map.hh.

5.70.2 Member Function Documentation

5.70.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 183 of file map.hh.

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

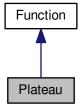
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.71 Plateau Class Reference

Plateau.

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

Inheritance diagram for Plateau:



Public Member Functions

• Plateau (int bv_size)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

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

5.71.1 Detailed Description

Plateau.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 239 of file theory.hh.

5.71.2 Member Function Documentation

```
5.71.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
_bv_size + 2
```

Reimplemented from Function.

Definition at line 263 of file theory.hh.

5.71.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 259 of file theory.hh.

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

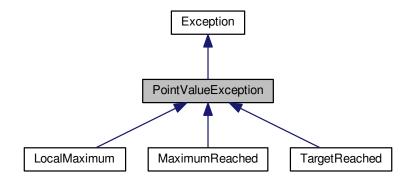
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.72 PointValueException Class Reference

Point-value exception.

#include <hnco/exception.hh>

Inheritance diagram for PointValueException:



Public Member Functions

- PointValueException (const point_value_t &pv)
 Constructor.
- const point_value_t & get_point_value () const Get point-value.

Protected Attributes

point_value_t _pvPoint-value.

5.72.1 Detailed Description

Point-value exception.

Definition at line 38 of file exception.hh.

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

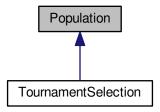
• lib/hnco/exception.hh

5.73 Population Class Reference

Population.

#include <hnco/algorithms/population.hh>

Inheritance diagram for Population:



Public Types

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

Public Member Functions

• Population (int population_size, int n)

Constructor.

• std::size_t size () const

• void random ()

Initialize the population with random bit vectors.

bit_vector_t & get_bv (int i)

Get a bit vector.

• const bit_vector_t & get_bv (int i) const

Get a bit vector.

Get sorted bit vectors

• const bit_vector_t & get_best_bv (int i) const

Get best bit vector.

• const bit_vector_t & get_best_bv () const

Get best bit vector.

const bit_vector_t & get_worst_bv (int i) const

Get worst bit vector.

Get sorted values

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

Evaluation and sorting

void eval (function::Function *function)

Evaluate the population.

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

Parallel evaluation of the population.

void sort ()

Sort the lookup table.

Selection

• void plus_selection (const Population &offsprings)

Plus selection.

void comma selection (const Population & offsprings)

Comma selection.

Protected Attributes

```
• std::vector < bit\_vector\_t > \_bvs
```

Bit vectors.

std::vector< index_value_t > _lookup

Lookup table.

• std::function< bool(const index_value_t &, const index_value_t &)> _compare_index_value Binary operator for comparing index-value pairs.

5.73.1 Detailed Description

Population.

Definition at line 36 of file population.hh.

5.73.2 Member Function Documentation

```
5.73.2.1 comma_selection()
```

Comma selection.

Precondition

Offspring population must be sorted.

Warning

The function does not break ties randomly as it should.

Definition at line 93 of file population.cc.

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

Get best bit vector.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 90 of file population.hh.

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

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

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 96 of file population.hh.

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

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

Get best value.

Parameters

i Index in the sorted population

Precondition

The population must be sorted.

Definition at line 119 of file population.hh.

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

Get best value.

Precondition

The population must be sorted.

Definition at line 125 of file population.hh.

```
5.73.2.6 get_worst_bv()
```

Get worst bit vector.

Parameters

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

Precondition

The population must be sorted.

Definition at line 104 of file population.hh.

5.73.2.7 plus_selection()

Plus selection.

Precondition

Both populations must be sorted.

Warning

The function does not break ties randomly as it should.

Definition at line 74 of file population.cc.

5.73.3 Member Data Documentation

5.73.3.1 _compare_index_value

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

Initial value:

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

Binary operator for comparing index-value pairs.

Definition at line 57 of file population.hh.

5.73.3.2 _lookup

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

Lookup table.

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

Definition at line 54 of file population.hh.

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

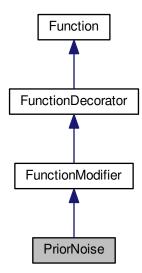
- · lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

5.74 PriorNoise Class Reference

Prior noise.

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

Inheritance diagram for PriorNoise:



Public Member Functions

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

Information about the function

• size_t get_bv_size ()

Get bit vector size.

• double get_maximum ()

Get the global maximum.

• bool has_known_maximum ()

Check for a known maximum.

• bool provides_incremental_evaluation ()

Check whether the function provides incremental evaluation.

Evaluation

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

Private Attributes

neighborhood::Neighborhood * _neighborhood
 Neighborhood.

bit_vector_t _noisy_bv

Noisy bit vector.

Additional Inherited Members

5.74.1 Detailed Description

Prior noise.

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

5.74.2 Member Function Documentation

5.74.2.1 get_maximum()

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

Get the global maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 67 of file prior-noise.hh.

5.74.2.2 has_known_maximum()

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

Check for a known maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 73 of file prior-noise.hh.

5.74.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Function.

Definition at line 77 of file prior-noise.hh.

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

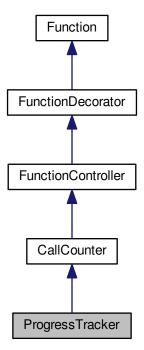
- · lib/hnco/functions/decorators/prior-noise.hh
- lib/hnco/functions/decorators/prior-noise.cc

5.75 ProgressTracker Class Reference

ProgressTracker.

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

Inheritance diagram for ProgressTracker:



Classes

struct Event

Event.

Public Member Functions

ProgressTracker (Function *function)

Constructor.

const Event & get_last_improvement ()

Get the last improvement.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

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

Update after a safe evaluation.

Setters

void set_log_improvement (bool x)

Log improvement.

void set_stream (std::ostream *x)

Output stream.

Protected Member Functions

void update_last_improvement (double value)

Update last improvement.

Protected Attributes

• Event _last_improvement

Last improvement.

Parameters

• bool <u>log_improvement</u> = false

Log improvement.

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

Output stream.

5.75.1 Detailed Description

ProgressTracker.

A ProgressTracker is a CallCounter which records the last event, that is the time and value of the last improvement.

Definition at line 212 of file function-controller.hh.

5.75.2 Member Function Documentation

Evaluate a bit vector.

Exceptions

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

5.75.2.2 get_last_improvement()

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

Get the last improvement.

Warning

If _last_improvement.time is zero then _function has never been called. The Event returned by get_last_← improvement has therefore no meaning.

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

5.75.2.3 incremental_eval()

Incremental evaluation.

Exceptions

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

5.75.2.4 update()

Update after a safe evaluation.

Exceptions

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

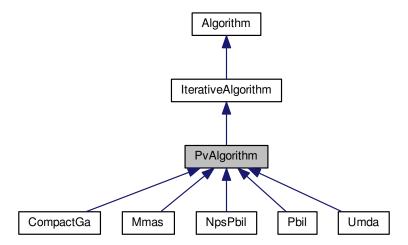
The documentation for this class was generated from the following files:

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

5.76 PvAlgorithm Class Reference

Probability vector algorithm.

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



Public Types

- enum { LOG PV, LOG ENTROPY, LAST_LOG }
- typedef std::bitset< LAST_LOG > log_flags_t

Public Member Functions

• PvAlgorithm (int n)

Constructor.

void set_log_flags (const log_flags_t &lf)

Set log flags.

Setters

void set_log_num_components (int x)
 Set the number of probability vector components to log.

Protected Member Functions

void log ()
 Log.

Protected Attributes

pv_t _pv

Probability vector.

• double _lower_bound

Lower bound of probability.

• double <u>upper_bound</u>

Upper bound of probability.

• log_flags_t _log_flags Log flags.

Parameters

• int _log_num_components = 5

Number of probability vector components to log.

5.76.1 Detailed Description

Probability vector algorithm.

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

5.76.2 Member Enumeration Documentation

5.76.2.1 anonymous enum

anonymous enum

5.77 Qubo Class Reference 167

Enumerator

LOG_PV	Log probability vector.
LOG_ENTROPY	Log entropy.

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

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

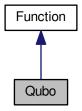
- lib/hnco/algorithms/pv/pv-algorithm.hh
- lib/hnco/algorithms/pv/pv-algorithm.cc

5.77 Qubo Class Reference

Quadratic unconstrained binary optimization.

#include <hnco/functions/qubo.hh>

Inheritance diagram for Qubo:



Public Member Functions

• Qubo ()

Constructor.

void load (std::istream &stream)

Load an instance.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Attributes

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

5.77.1 Detailed Description

Quadratic unconstrained binary optimization.

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

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

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

There are some differences between WalshExpansion2 and Qubo:

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

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

References:

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

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

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

Definition at line 74 of file qubo.hh.

5.77.2 Member Function Documentation

```
5.77.2.1 load()
```

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

Load an instance.

Exceptions

Error

Definition at line 35 of file qubo.cc.

5.77.3 Member Data Documentation

5.77.3.1 _q

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

Matrix.

n x n upper triangular matrix.

Definition at line 83 of file qubo.hh.

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

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

5.78 Random Struct Reference

Random numbers.

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

Static Public Member Functions

• static double uniform ()

Next uniformly distributed sample.

• static double normal ()

Next normally distributed sample.

static bool random_bit ()

Next random bit.

Static Public Attributes

 static std::mt19937 engine Engine.

5.78.1 Detailed Description

Random numbers.

Definition at line 33 of file random.hh.

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

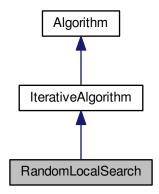
- lib/hnco/random.hh
- · lib/hnco/random.cc

5.79 RandomLocalSearch Class Reference

Random local search.

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

Inheritance diagram for RandomLocalSearch:



Public Member Functions

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

Constructor.

· void init ()

Random initialization.

void init (const bit_vector_t &x)

Explicit initialization.

• void init (const bit_vector_t &x, double value)

Explicit initialization.

• const point_value_t & get_solution ()

Solution.

Setters

void set_compare (std::function< bool(double, double)> x)

Set the binary operator for comparing evaluations.

void set_patience (int x)

Set patience.

• void set_incremental_evaluation (bool x)

Set incremental evaluation.

Protected Member Functions

• void iterate ()

Single iteration.

• void iterate_full ()

Single iteration with full evaluation.

• void iterate_incremental ()

Single iteration with incremental evaluation.

Protected Attributes

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

Neighborhood.

· int _num_failures

Number of failure.

Parameters

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

Patience.

• bool _incremental_evaluation = false

Incremental evaluation.

5.79.1 Detailed Description

Random local search.

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

5.79.2 Member Function Documentation

5.79.2.1 set_patience()

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

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximum exception

Parameters

x Patience

If $x \le 0$ then patience is considered infinite, meaning that the algorithm will never throw any LocalMaximum exception.

Definition at line 110 of file random-local-search.hh.

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

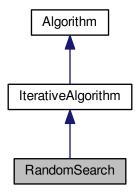
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

5.80 RandomSearch Class Reference

Random search.

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

Inheritance diagram for RandomSearch:



Public Member Functions

• RandomSearch (int n) Constructor.

Protected Member Functions

• void iterate ()

Single iteration.

Private Attributes

bit_vector_t _candidate
 Candidate.

Additional Inherited Members

5.80.1 Detailed Description

Random search.

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

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

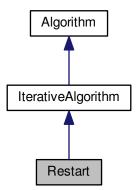
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

5.81 Restart Class Reference

Restart.

#include <hnco/algorithms/restart.hh>

Inheritance diagram for Restart:



Public Member Functions

• Restart (int n, Algorithm *algorithm)

Constructor.

void set_function (function::Function *function)

Set function

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

Set functions.

Private Member Functions

• void iterate ()

Optimize.

Private Attributes

Algorithm * _algorithm
 Algorithm.

Additional Inherited Members

5.81.1 Detailed Description

Restart.

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

Definition at line 38 of file restart.hh.

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

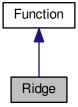
- · lib/hnco/algorithms/restart.hh
- · lib/hnco/algorithms/restart.cc

5.82 Ridge Class Reference

Ridge.

#include <hnco/functions/theory.hh>

Inheritance diagram for Ridge:



Public Member Functions

```
• Ridge (int bv_size)
```

Constructor.

size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size
 Bit vector size.

5.82.1 Detailed Description

Ridge.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 203 of file theory.hh.

5.82.2 Member Function Documentation

```
5.82.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
2 * _bv_size
```

Reimplemented from Function.

Definition at line 227 of file theory.hh.

5.82.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 223 of file theory.hh.

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

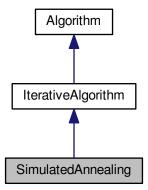
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.83 SimulatedAnnealing Class Reference

Simulated annealing.

#include <hnco/algorithms/ls/simulated-annealing.hh>

Inheritance diagram for SimulatedAnnealing:



Public Member Functions

• SimulatedAnnealing (int n, neighborhood::Neighborhood *neighborhood)

Constructor.

• void init ()

Initialization.

Setters

void set_num_transitions (int x)

Set the number of accepted transitions before annealing.

• void set_num_trials (int x)

Set the Number of trials.

void set_initial_acceptance_probability (double x)

Set the initial acceptance probability.

void set_beta_ratio (double x)

Set ratio for beta.

Private Member Functions

· void init_beta ()

Initialize beta.

· void iterate ()

Single iteration.

Private Attributes

• neighborhood::Neighborhood * _neighborhood

Neighborhood.

• double _beta

Inverse temperature.

double _current_value

Current value.

· int transitions

Number of accepted transitions.

Parameters

• int _num_transitions = 50

Number of accepted transitions before annealing.

• int _num_trials = 100

Number of trials.

• double _initial_acceptance_probability = 0.6

Initial acceptance probability.

• double beta ratio = 1.2

Ratio for beta.

Additional Inherited Members

5.83.1 Detailed Description

Simulated annealing.

Reference:

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. Science 220, 4598 (May 1983), 671–680.

Definition at line 44 of file simulated-annealing.hh.

5.83.2 Member Function Documentation

5.83.2.1 init_beta()

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

Initialize beta.

Requires (2 * _num_trials) evaluations. This should be taken into account when using OnBudgetFunction.

Definition at line 34 of file simulated-annealing.cc.

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

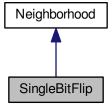
- · lib/hnco/algorithms/ls/simulated-annealing.hh
- · lib/hnco/algorithms/ls/simulated-annealing.cc

5.84 SingleBitFlip Class Reference

One bit neighborhood.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for SingleBitFlip:



Public Member Functions

SingleBitFlip (int n)
 Constructor.

Private Member Functions

void sample_bits ()
 Sample bits.

Additional Inherited Members

5.84.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

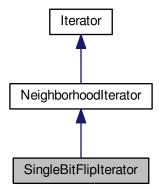
• lib/hnco/neighborhoods/neighborhood.hh

5.85 SingleBitFlipIterator Class Reference

Single bit flip neighborhood iterator.

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

Inheritance diagram for SingleBitFlipIterator:



Public Member Functions

```
    SingleBitFlipIterator (int n)
```

Constructor.

• bool has_next ()

Has next bit vector.

• const bit_vector_t & next ()

Next bit vector.

Private Attributes

```
    size_t _index
    Index of the last flipped bit.
```

Additional Inherited Members

5.85.1 Detailed Description

Single bit flip neighborhood iterator.

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

5.85.2 Constructor & Destructor Documentation

5.85.2.1 SingleBitFlipIterator()

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

Constructor.

Parameters

```
n Size of bit vectors
```

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

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

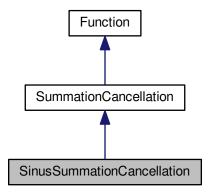
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.86 SinusSummationCancellation Class Reference

Summation cancellation with sinus.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SinusSummationCancellation:



Public Member Functions

• SinusSummationCancellation (int n)

Constructor.

double eval (const bit_vector_t &x)

Evaluate a bit vector.

Additional Inherited Members

5.86.1 Detailed Description

Summation cancellation with sinus.

Reference:

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

Definition at line 103 of file cancellation.hh.

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

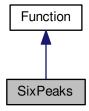
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.87 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



Public Member Functions

• SixPeaks (int bv_size, int threshold)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

size_t _bv_size

Bit vector size.

• int _threshold

Threshold.

• int _maximum

Maximum.

5.87.1 Detailed Description

Six Peaks.

It is defined by

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

where:

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

This function has six maxima, of which exactly four are global ones.

For example, if n = 6 and t = 1:

- f(111111) = 6 (local maximum)
- f(1111110) = 5
- f(111100) = 10 (global maximum)

Reference:

J. S. De Bonet, C. L. Isbell, and P. Viola. 1996. MIMIC: finding optima by estimating probability densities. In Advances in Neural Information Processing Systems. Vol. 9. MIT Press, Denver.

Definition at line 128 of file four-peaks.hh.

5.87.2 Member Function Documentation

```
5.87.2.1 get_maximum()
```

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

Get the global maximum.

Returns

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

Reimplemented from Function.

Definition at line 159 of file four-peaks.hh.

5.87.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 155 of file four-peaks.hh.

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

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

5.88 SpinHerding Class Reference

Herding with spin variables.

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

Public Types

```
enum {
    SAMPLE_GREEDY, SAMPLE_RLS, SAMPLE_DLS, SAMPLE_NN,
    LAST_SAMPLE }
```

Public Member Functions

• SpinHerding (int n)

Constructor.

• void init ()

Initialization.

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

Sample a bit vector.

• double error (const SpinMoment &target)

Compute the error.

• double delta (const SpinMoment &target)

Compute the norm of the moment increment.

Setters

• void set randomize bit order (bool x)

Randomize bit order.

void set_sampling_method (int x)

Set the sampling method.

void set_num_seq_updates (int x)

Set the number of sequential updates per sample.

void set_num_par_updates (int x)

Set the number of parallel updates per sample.

void set_weight (double x)

Set the weight of second order moments.

Protected Member Functions

void compute_delta (const SpinMoment &target)

Compute delta.

void sample_greedy (bit_vector_t &x)

Sample by means of a greedy algorithm.

double q_derivative (const bit_vector_t &x, size_t i)

Derivative of q.

double q_variation (const bit_vector_t &x, size_t i)

Variation of q.

void sample_rls (bit_vector_t &x)

Sample by means of random local search.

void sample_dls (bit_vector_t &x)

Sample by means of deterministic local search.

void sample_nn (bit_vector_t &x)

Sample by means of a neural network.

void update_counters (const bit_vector_t &x)

Update counters.

Protected Attributes

· SpinMoment _delta

Delta moment.

SpinMoment _count

Counter moment.

bit_vector_t _state

State.

permutation_t _permutation

Permutation.

std::uniform_int_distribution< int > _choose_bit

Choose bit.

· int _time

Time.

Parameters

• bool <u>_randomize_bit_order</u> = false

Randomize bit order.

int _sampling_method = SAMPLE_GREEDY

Sampling method.

· int num seq updates

Number of sequential updates per sample.

• int _num_par_updates = 1

Number of parallel updates per sample.

• double _weight = 1

Weight of second order moments.

5.88.1 Detailed Description

Herding with spin variables.

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

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

5.88.2 Member Enumeration Documentation

5.88.2.1 anonymous enum

anonymous enum

Enumerator

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

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

5.88.3 Constructor & Destructor Documentation

5.88.3.1 SpinHerding()

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

Constructor.

Parameters

```
n Size of bit vectors
```

_num_seq_updates is initialized to n.

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

5.88.4 Member Function Documentation

5.88.4.1 q_variation()

Variation of q.

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

Definition at line 157 of file herding-spin.cc.

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

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

5.89 SpinMoment Struct Reference

Moment for spin variables.

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

Public Member Functions

• SpinMoment (int n)

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

· void init ()

Initialize accumulators.

void add (const bit_vector_t &x)

Update accumulators.

void average (int count)

Compute average.

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

Update moment.

void bound (double margin)

Bound moment.

· double distance (const SpinMoment &p) const

Distance.

• double norm_2 () const

Compute the norm 2.

• double diameter () const

Compute the diameter.

· size_t size () const

Size.

Public Attributes

std::vector< double > _first

First moment.

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

Second moment.

• double _weight = 1

Weight of second order moments.

5.89.1 Detailed Description

Moment for spin variables.

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

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

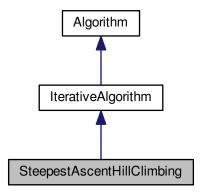
- lib/hnco/algorithms/hea/moment-spin.hh
- lib/hnco/algorithms/hea/moment-spin.cc

5.90 SteepestAscentHillClimbing Class Reference

Steepest ascent hill climbing.

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

Inheritance diagram for SteepestAscentHillClimbing:



Public Member Functions

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

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.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

std::vector < bit_vector_t > _candidates
 Potential candidate.

neighborhood::NeighborhoodIterator * _neighborhood
 Neighborhood.

5.90.1 Detailed Description

Steepest ascent hill climbing.

Definition at line 39 of file steepest-ascent-hill-climbing.hh.

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

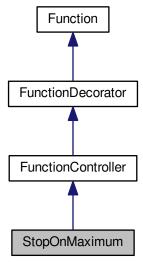
- · lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

5.91 StopOnMaximum Class Reference

Stop on maximum.

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

Inheritance diagram for StopOnMaximum:



Public Member Functions

• StopOnMaximum (Function *function)

Constructor.

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

void update (const bit_vector_t &x, double value)

Update after a safe evaluation.

Additional Inherited Members

5.91.1 Detailed Description

Stop on maximum.

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

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

5.91.2 Constructor & Destructor Documentation

5.91.2.1 StopOnMaximum()

Constructor.

Parameters

```
function Decorated function
```

Precondition

function->has_known_maximum()

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

5.91.3 Member Function Documentation

Evaluate a bit vector.

Exceptions

MaximumReached

Implements Function.

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

5.91.3.2 incremental_eval()

Incremental evaluation.

Exceptions

MaximumReached

Reimplemented from Function.

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

5.91.3.3 update()

Update after a safe evaluation.

Exceptions

MaximumReached

Reimplemented from Function.

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

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

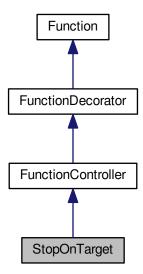
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.92 StopOnTarget Class Reference

Stop on target.

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

Inheritance diagram for StopOnTarget:



Public Member Functions

Constructor.

• StopOnTarget (Function *function, double target)

Evaluation

double eval (const bit_vector_t &)

Evaluate a bit vector.

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

Incremental evaluation.

void update (const bit_vector_t &x, double value)
 Update after a safe evaluation.

Private Attributes

• double _target Target.

Additional Inherited Members

5.92.1 Detailed Description

Stop on target.

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

5.92.2 Constructor & Destructor Documentation

5.92.2.1 StopOnTarget()

Constructor.

Parameters

```
function Decorated function
```

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

5.92.3 Member Function Documentation

Evaluate a bit vector.

Exceptions

TargetReached

Implements Function.

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

5.92.3.2 incremental_eval()

Incremental evaluation.

Exceptions

TargetReached

Reimplemented from Function.

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

5.92.3.3 update()

Update after a safe evaluation.

Exceptions

TargetReached

Reimplemented from Function.

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

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

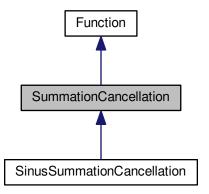
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

5.93 SummationCancellation Class Reference

Summation cancellation.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SummationCancellation:



Public Member Functions

• SummationCancellation (int n)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

double eval (const bit_vector_t &x)

Evaluate a bit vector.

• bool has_known_maximum ()

Check for a known maximum.

double get_maximum ()

Get the global maximum.

Protected Member Functions

void convert (const bit_vector_t &x)

Convert a bit vector into a real vector.

Protected Attributes

size_t _bv_size

Bit vector size.

• $std::vector < double > _buffer$

Buffer.

5.93.1 Detailed Description

Summation cancellation.

Encoding of a signed integer:

- bit 0: sign
- bits 1 to 8: two's complement representation

Reference:

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

Definition at line 47 of file cancellation.hh.

5.93.2 Constructor & Destructor Documentation

5.93.2.1 SummationCancellation()

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

Constructor.

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

Parameters

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

Definition at line 70 of file cancellation.hh.

5.93.3 Member Function Documentation

5.93.3.1 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 86 of file cancellation.hh.

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

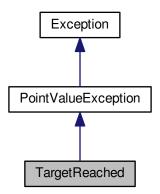
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.94 TargetReached Class Reference

target reached

#include <hnco/exception.hh>

Inheritance diagram for TargetReached:



Public Member Functions

TargetReached (const point_value_t &pv)
 Constructor.

Additional Inherited Members

5.94.1 Detailed Description

target reached

Definition at line 61 of file exception.hh.

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

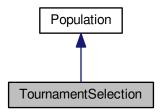
lib/hnco/exception.hh

5.95 TournamentSelection Class Reference

Population with tournament selection.

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

Inheritance diagram for TournamentSelection:



Public Member Functions

• TournamentSelection (int population_size, int n)

Constructor.

const bit_vector_t & select ()
 Selection.

Setters

void set_tournament_size (int x)
 Set the tournament size.

Private Attributes

 std::uniform_int_distribution < int > _choose_individual Random index.

Parameters

• int _tournament_size = 10 Tournament size.

Additional Inherited Members

5.95.1 Detailed Description

Population with tournament selection.

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

5.95.2 Member Function Documentation

5.95.2.1 select()

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

Selection.

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

Precondition

The population must be evaluated.

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

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

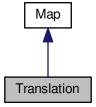
- lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

5.96 Translation Class Reference

Translation.

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

Inheritance diagram for Translation:



Public Member Functions

```
void random (int n)
    Random instance.
void map (const bit_vector_t &input, bit_vector_t &output)
    Map.
size_t get_input_size ()
    Get input size.
size_t get_output_size ()
    Get output size.
bool is_surjective ()
```

Private Member Functions

Check for surjective map.

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

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

Load.
```

Private Attributes

bit_vector_t _bv
 Translation vector.

Friends

· class boost::serialization::access

5.96.1 Detailed Description

Translation.

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

Definition at line 70 of file map.hh.

5.96.2 Member Function Documentation

```
5.96.2.1 is_surjective()
```

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 121 of file map.hh.

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

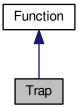
- · lib/hnco/map.hh
- · lib/hnco/map.cc

5.97 Trap Class Reference

Trap.

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

Inheritance diagram for Trap:



Public Member Functions

Trap (int bv_size, int num_traps)

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

bool has_known_maximum ()

Check for a known maximum.

• double get_maximum ()

Get the global maximum.

Private Attributes

```
    size_t _bv_size
        Bit vector size.
    int _num_traps
        Number of traps.
    int _trap_size
```

Trap size.

5.97.1 Detailed Description

Trap.

Reference:

Kalyanmoy Deb and David E. Goldberg. 1993. Analyzing Deception in Trap Functions. In Foundations of Genetic Algorithms 2, L. Darrell Whitley (Ed.). Morgan Kaufmann, San Mateo, CA, 93–108.

Definition at line 43 of file trap.hh.

5.97.2 Constructor & Destructor Documentation

Constructor.

Parameters

bv_size	Bit vector size
num_traps	Number of traps

Warning

bv_size must be a multiple of num_traps

Definition at line 64 of file trap.hh.

5.97.3 Member Function Documentation

5.98 Umda Class Reference 203

5.97.3.1 get_maximum() double get_maximum () [inline], [virtual] Get the global maximum. Returns _bv_size Reimplemented from Function. Definition at line 88 of file trap.hh. 5.97.3.2 has_known_maximum() bool has_known_maximum () [inline], [virtual] Check for a known maximum. Returns true Reimplemented from Function. Definition at line 84 of file trap.hh. The documentation for this class was generated from the following files:

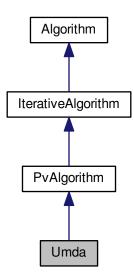
- lib/hnco/functions/trap.hh
- lib/hnco/functions/trap.cc

5.98 Umda Class Reference

Univariate marginal distribution algorithm.

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

Inheritance diagram for Umda:



Public Member Functions

- Umda (int n, int population_size)

 Constructor.
- void init ()

 Initialization.

Setters

• void set_selection_size (int x)

Set the selection size.

Protected Member Functions

• void iterate ()

Single iteration.

Protected Attributes

• Population _population

Population.

pv_t _mean

Mean of selected bit vectors.

Parameters

• int _selection_size = 1 Selection size.

Additional Inherited Members

5.98.1 Detailed Description

Univariate marginal distribution algorithm.

Reference:

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

Definition at line 40 of file umda.hh.

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

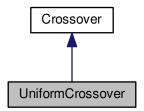
- · lib/hnco/algorithms/pv/umda.hh
- · lib/hnco/algorithms/pv/umda.cc

5.99 UniformCrossover Class Reference

Uniform crossover.

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

Inheritance diagram for UniformCrossover:



Public Member Functions

void breed (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)
 Breed.

5.99.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

5.99.2 Member Function Documentation

5.99.2.1 breed()

Breed.

The offspring is the uniform crossover of two parents.

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implements Crossover.

Definition at line 30 of file crossover.cc.

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

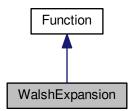
- · lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

5.100 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



Public Member Functions

• WalshExpansion ()

Constructor.

size_t get_bv_size ()

Get bit vector size.

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

Random instance.

double eval (const bit_vector_t &)

Evaluate a bit vector.

· void display (std::ostream &stream)

Display

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

Set terms.

Private Member Functions

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

Private Attributes

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

Friends

· class boost::serialization::access

5.100.1 Detailed Description

Walsh expansion.

Its expression is of the form

$$f(x) = \sum_{u} a_u (-1)^{x \cdot u}$$

where the sum is over a subset of $\{0,1\}^n$ and $x \cdot u = \sum_i x_i u_i$ is mod 2. The real numbers a_u are the coefficients of the expansion and the bit vectors u are its feature vectors.

Definition at line 53 of file walsh-expansion.hh.

5.100.2 Member Function Documentation

5.100.2.1 random()

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

Random instance.

Parameters

n	Size of bit vector
num_features	Number of feature vectors
stddev	Standard deviation of the coefficients

Definition at line 34 of file walsh-expansion.cc.

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

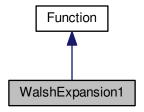
- · lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

5.101 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

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

Inheritance diagram for WalshExpansion1:



Public Member Functions

WalshExpansion1 ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

• void random (int n, double stddev)

Random instance.

double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Member Functions

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

Private Attributes

std::vector< double > _linear
 Linear part.

Friends

· class boost::serialization::access

5.101.1 Detailed Description

Walsh expansion of degree 1.

Its expression is of the form

$$f(x) = \sum_{i} a_i (1 - 2x_i)$$

or equivalently

$$f(x) = \sum_{i} a_i (-1)^{x_i}$$

Definition at line 50 of file walsh-expansion-1.hh.

5.101.2 Member Function Documentation

5.101.2.1 random()

Random instance.

Parameters

n	Size of bit vector
stddev	Standard deviation of the coefficients

Definition at line 33 of file walsh-expansion-1.cc.

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

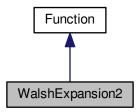
- · lib/hnco/functions/walsh/walsh-expansion-1.hh
- lib/hnco/functions/walsh/walsh-expansion-1.cc

5.102 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

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

Inheritance diagram for WalshExpansion2:



Public Member Functions

• WalshExpansion2 ()

Constructor.

• size_t get_bv_size ()

Get bit vector size.

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

Random instance.

• double eval (const bit_vector_t &)

Evaluate a bit vector.

Private Member Functions

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

Private Attributes

std::vector< double > _linear
 Linear part.

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

Quadratic part.

Friends

· class boost::serialization::access

5.102.1 Detailed Description

Walsh expansion of degree 2.

Its expression is of the form

$$f(x) = \sum_{i} a_i (1 - 2x_i) + \sum_{i < j} a_{ij} (1 - 2x_i) (1 - 2x_j)$$

or equivalently

$$f(x) = \sum_{i} a_i (-1)^{x_i} + \sum_{i < j} a_{ij} (-1)^{x_i + x_j}$$

where the sum $x_i + x_j$ is mod 2 (xor).

Definition at line 52 of file walsh-expansion-2.hh.

5.102.2 Member Function Documentation

5.102.2.1 random()

Random instance.

Parameters

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

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

5.102.3 Member Data Documentation

5.102.3.1 _quadratic

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

Quadratic part.

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

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

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

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

5.103 Function::WalshTransformTerm Struct Reference

Walsh transform term.

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

Public Member Functions

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

Public Attributes

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

Coefficient.

5.103.1 Detailed Description

Walsh transform term.

Definition at line 44 of file function.hh.

5.103.2 Member Data Documentation

5.103.2.1 feature

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

Feature.

Implemented with a vector bool instead of a bit_vector_t to reduce the memory consumption.

Definition at line 51 of file function.hh.

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

· lib/hnco/functions/function.hh

Index

_compare_index_value	hnco, 19
hnco::algorithm::Population, 159	Cache, 45
_expression	hnco::function::Cache, 46
hnco::function::MaxSat, 115	CallCounter, 47
_functions	comma_selection
hnco::algorithm::Algorithm, 33	hnco::algorithm::Population, 156
_lookup	CompactGa, 48
hnco::algorithm::Population, 159	CompleteSearch, 50
_q	compute_walsh_transform
hnco::function::Qubo, 169	hnco::function::Function, 65
_quadratic	Crossover, 51
hnco::function::WalshExpansion2, 211	010350761, 31
AdditiveGaussianNoise, 27	DeceptiveJump, 52
AffineMap, 29	EqualDraduate 54
Algorithm, 31	EqualProducts, 54
•	Error, 56
bernoulli_trials	eval
hnco::neighborhood::MultiBitFlip, 122	hnco::function::OnBudgetFunction, 140
BernoulliProcess, 34	hnco::function::ProgressTracker, 164
hnco::neighborhood::BernoulliProcess, 35, 36	hnco::function::StopOnMaximum, 191
BiasedCrossover, 37	hnco::function::StopOnTarget, 193
BinaryHerding, 38	Exception, 57
BinaryMoment, 40	Factorization, 58
bit_t	hnco::function::Factorization, 59
hnco, 14	feature
bm_add_rows	hnco::function::Function::WalshTransformTerm,
hnco, 15	212
bm_identity	FirstAscentHillClimbing, 60
hnco, 15	FourPeaks, 61
bm_invert	Function, 64
hnco, 16	Function::WalshTransformTerm, 212
bm_multiply	FunctionController, 68
hnco, 16	FunctionDecorator, 69
bm_rank	FunctionMapComposition, 70
hnco, 17	hnco::function::FunctionMapComposition, 71
bm_row_echelon_form	FunctionModifier, 73
hnco, 17	FunctionPlugin, 74
bm_solve	hnco::function::FunctionPlugin, 75
hnco, 17	
bm_solve_upper_triangular	GeneticAlgorithm, 75
hnco, 18	hnco::algorithm::GeneticAlgorithm, 77
BmPbil, 41	get_best_bv
breed	hnco::algorithm::Population, 156, 157
hnco::algorithm::BiasedCrossover, 37	get_best_value
hnco::algorithm::Crossover, 52	hnco::algorithm::Population, 157
hnco::algorithm::UniformCrossover, 206	get_last_improvement
bv_from_vector_bool	hnco::function::ProgressTracker, 164
hnco, 19	get_maximum
by to vector bool	hnco::function::AdditiveGaussianNoise, 28

hnco::function::DeceptiveJump, 53	sbm_multiply, 19
hnco::function::FourPeaks, 63	sparse_bit_matrix_t, 14
hnco::function::Function, 66	sparse_bit_vector_t, 15
hnco::function::FunctionMapComposition, 72	hnco::AffineMap
hnco::function::Hiff, 90	is_surjective, 30
hnco::function::Jump, 98	random, 30
hnco::function::LeadingOnes, 101	hnco::LinearMap
hnco::function::Needle, 127	is_surjective, 105
hnco::function::Negation, 129	random, 105
hnco::function::OneMax, 142	
	hnco::Map
hnco::function::Plateau, 153	is_surjective, 110
hnco::function::PriorNoise, 161	hnco::MapComposition
hnco::function::Ridge, 175	is_surjective, 112
hnco::function::SixPeaks, 183	MapComposition, 111
hnco::function::Trap, 202	hnco::Permutation
get_worst_bv	is_surjective, 151
hnco::algorithm::Population, 158	hnco::Translation
	is_surjective, 200
HammingBall, 78	hnco::algorithm, 20
hnco::neighborhood::HammingBall, 79	_
HammingSphere, 80	hnco::algorithm::Algorithm
hnco::neighborhood::HammingSphere, 81	_functions, 33
HammingSphereIterator, 81	set_solution, 33
hnco::neighborhood::HammingSphereIterator, 83	update_solution, 33
has known maximum	hnco::algorithm::BiasedCrossover
hnco::function::AdditiveGaussianNoise, 28	breed, 37
	hnco::algorithm::Crossover
hnco::function::DeceptiveJump, 53	breed, 52
hnco::function::FourPeaks, 63	hnco::algorithm::GeneticAlgorithm
hnco::function::FunctionMapComposition, 72	GeneticAlgorithm, 77
hnco::function::Hiff, 90	set_allow_stay, 78
hnco::function::Jump, 98	hnco::algorithm::IterativeAlgorithm
hnco::function::LeadingOnes, 101	
hnco::function::LinearFunction, 103	IterativeAlgorithm, 94
hnco::function::Needle, 127	maximize, 94
hnco::function::Negation, 129	set_num_iterations, 95
hnco::function::OneMax, 142	hnco::algorithm::MuCommaLambdaEa
hnco::function::Plateau, 153	MuCommaLambdaEa, 120
hnco::function::PriorNoise, 161	set_allow_stay, 121
hnco::function::Ridge, 175	hnco::algorithm::MuPlusLambdaEa
hnco::function::SixPeaks, 183	MuPlusLambdaEa, 125
	set allow stay, 125
hnco::function::SummationCancellation, 196	hnco::algorithm::OnePlusLambdaCommaLambdaGa
hnco::function::Trap, 203	OnePlusLambdaCommaLambdaGa, 145
Hboa, 83	
Hea	hnco::algorithm::OnePlusOneEa
hnco::algorithm::hea::Hea, 88	OnePlusOneEa, 147
Hea< Moment, Herding >, 84	set_allow_stay, 147
Hiff, 89	set_num_iterations, 147
hnco, 11	hnco::algorithm::Population
bit_t, 14	_compare_index_value, 159
bm_add_rows, 15	_lookup, 159
bm_identity, 15	comma_selection, 156
bm_invert, 16	get_best_bv, 156, 157
bm_multiply, 16	get_best_value, 157
_ , ,	
bm_rank, 17	get_worst_bv, 158
bm_row_echelon_form, 17	plus_selection, 158
bm_solve, 17	hnco::algorithm::RandomLocalSearch
bm_solve_upper_triangular, 18	set_patience, 171
bv_from_vector_bool, 19	hnco::algorithm::SimulatedAnnealing
bv_to_vector_bool, 19	init_beta, 178

hnco::algorithm::TournamentSelection select, 199	hnco::function::LinearFunction has_known_maximum, 103
hnco::algorithm::UniformCrossover	random, 103
breed, 206	hnco::function::MaxSat
hnco::algorithm::bm_pbil, 22	_expression, 115
hnco::algorithm::bm_pbil::BmPbil	load, 114
set_selection_size, 44	random, 114, 115
hnco::algorithm::hea, 22	hnco::function::Needle
hnco::algorithm::hea::Hea	get maximum, 127
Hea, 88	has_known_maximum, 127
set reset period, 88	hnco::function::Negation
set_selection_size, 88	get_maximum, 129
hnco::algorithm::hea::SpinHerding	has_known_maximum, 129
q_variation, 186	provides_incremental_evaluation, 129
SpinHerding, 186	hnco::function::NkLandscape
hnco::exception, 23	random, 136
hnco::function, 23	hnco::function::OnBudgetFunction
hnco::function::AdditiveGaussianNoise	eval, 140
get_maximum, 28	incremental_eval, 140
has_known_maximum, 28	update, 140
hnco::function::Cache	hnco::function::OneMax
Cache, 46	get_maximum, 142
provides_incremental_evaluation, 46	has_known_maximum, 142
hnco::function::DeceptiveJump	provides_incremental_evaluation, 143
get_maximum, 53	hnco::function::Plateau
has_known_maximum, 53	get_maximum, 153
hnco::function::EqualProducts	has_known_maximum, 153
random, 55	hnco::function::PriorNoise
hnco::function::Factorization	get_maximum, 161
Factorization, 59	has_known_maximum, 161
hnco::function::FourPeaks	provides_incremental_evaluation, 161
get_maximum, 63	hnco::function::ProgressTracker
has_known_maximum, 63	eval, 164
hnco::function::Function	get_last_improvement, 164
compute_walsh_transform, 65	incremental_eval, 164
get_maximum, 66	update, 165
incremental eval, 66	hnco::function::Qubo
provides_incremental_evaluation, 67	_q, 169
safe eval, 67	load, 168
hnco::function::WalshTransformTerm	hnco::function::Ridge
feature, 212	get_maximum, 175
hnco::function::FunctionController	- -
	has_known_maximum, 175
provides_incremental_evaluation, 69	hnco::function::SixPeaks
hnco::function::FunctionMapComposition	get_maximum, 183
FunctionMapComposition, 71	has_known_maximum, 183
get_maximum, 72	hnco::function::StopOnMaximum
has_known_maximum, 72	eval, 191
hnco::function::FunctionPlugin	incremental_eval, 191
FunctionPlugin, 75	StopOnMaximum, 190
hnco::function::Hiff	update, 191
get_maximum, 90	hnco::function::StopOnTarget
has_known_maximum, 90	eval, 193
hnco::function::Jump	incremental_eval, 194
get_maximum, 98	StopOnTarget, 193
has_known_maximum, 98	update, 194
hnco::function::LeadingOnes	hnco::function::SummationCancellation
get_maximum, 101	has_known_maximum, 196
has_known_maximum, 101	SummationCancellation, 196

hnco::function::Trap	Labs, 98
get_maximum, 202	LastEvaluation, 99
has_known_maximum, 203	LeadingOnes, 100
Trap, 202	LinearFunction, 102
hnco::function::WalshExpansion	LinearMap, 104
random, 207	load
hnco::function::WalshExpansion1	hnco::function::MaxSat, 114
random, 209	hnco::function::Qubo, 168
hnco::function::WalshExpansion2	LocalMaximum, 106
_quadratic, 211	LongPath, 107
random, 211	Ltga, 108
hnco::neighborhood, 25	
hnco::neighborhood::BernoulliProcess	Map, 109
BernoulliProcess, 35, 36	map
set_allow_stay, 36	hnco::neighborhood::Neighborhood, 132
set_probability, 36	MapComposition, 110
hnco::neighborhood::HammingBall	hnco::MapComposition, 111
HammingBall, 79	MaxSat, 113
hnco::neighborhood::HammingSphere	maximize
HammingSphere, 81	hnco::algorithm::IterativeAlgorithm, 94
hnco::neighborhood::HammingSphereIterator	MaximumReached, 112
HammingSphereIterator, 83	Mmas, 116
hnco::neighborhood::MultiBitFlip	Model, 117
bernoulli_trials, 122	ModelParameters, 118
MultiBitFlip, 122	MuCommaLambdaEa, 119
reservoir_sampling, 123	hnco::algorithm::MuCommaLambdaEa, 120
hnco::neighborhood::Neighborhood	MuPlusLambdaEa, 123
map, 132	hnco::algorithm::MuPlusLambdaEa, 125
mutate, 133	MultiBitFlip, 121
Neighborhood, 132	hnco::neighborhood::MultiBitFlip, 122
hnco::neighborhood::NeighborhoodIterator	mutate
NeighborhoodIterator, 134	hnco::neighborhood::Neighborhood, 133
hnco::neighborhood::SingleBitFlipIterator	
SingleBitFlipIterator, 180	Needle, 126
hnco::random, 26	Negation, 128
HncoEvaluator, 91	Neighborhood, 130
Hypercubelterator, 92	hnco::neighborhood::Neighborhood, 132
Tiyporoabonorator, oz	NeighborhoodIterator, 133
incremental_eval	hnco::neighborhood::NeighborhoodIterator, 134
hnco::function::Function, 66	NkLandscape, 134
hnco::function::OnBudgetFunction, 140	NpsPbil, 136
hnco::function::ProgressTracker, 164	O.D. I. IE. II. 100
hnco::function::StopOnMaximum, 191	OnBudgetFunction, 138
hnco::function::StopOnTarget, 194	OneMax, 141
init beta	OnePlusLambdaCommaLambdaGa, 143
hnco::algorithm::SimulatedAnnealing, 178	hnco::algorithm::OnePlusLambdaComma
is_surjective	LambdaGa, 145
hnco::AffineMap, 30	OnePlusOneEa, 145
hnco::LinearMap, 105	hnco::algorithm::OnePlusOneEa, 147
hnco::Map, 110	Devemptor Leas Deputation Duramid 140
hnco::MapComposition, 112	ParameterLessPopulationPyramid, 148
hnco::Permutation, 151	Pbil, 149
hnco::Translation, 200	Permutation, 150
IterativeAlgorithm, 93	Plateau, 152
hnco::algorithm::IterativeAlgorithm, 94	plus_selection
-	hnco::algorithm::Population, 158
Iterator, 95	PointValueException, 154 Population, 155
lump 97	PriorNoise, 160
Jump, 97	1 1101110136, 100

ProgressTracker, 162	SimulatedAnnealing, 176
ProgressTracker::Event, 57	SingleBitFlip, 178
provides_incremental_evaluation	SingleBitFlipIterator, 179
hnco::function::Cache, 46	hnco::neighborhood::SingleBitFlipIterator, 180
hnco::function::Function, 67	SinusSummationCancellation, 181
hnco::function::FunctionController, 69	SixPeaks, 182
hnco::function::Negation, 129	sparse_bit_matrix_t
hnco::function::OneMax, 143	hnco, 14
hnco::function::PriorNoise, 161	sparse_bit_vector_t
PvAlgorithm, 165	hnco, 15
	SpinHerding, 184
q_variation	hnco::algorithm::hea::SpinHerding, 186
hnco::algorithm::hea::SpinHerding, 186	SpinMoment, 187
Qubo, 167	SteepestAscentHillClimbing, 188
D 1 400	StopOnMaximum, 189
Random, 169	hnco::function::StopOnMaximum, 190
random	StopOnTarget, 192
hnco::AffineMap, 30	hnco::function::StopOnTarget, 193
hnco::LinearMap, 105	SummationCancellation, 194
hnco::function::EqualProducts, 55	hnco::function::SummationCancellation, 196
hnco::function::LinearFunction, 103	
hnco::function::MaxSat, 114, 115	TargetReached, 197
hnco::function::NkLandscape, 136	TournamentSelection, 198
hnco::function::WalshExpansion, 207	Translation, 199
hnco::function::WalshExpansion1, 209	Trap, 201
hnco::function::WalshExpansion2, 211	hnco::function::Trap, 202
RandomLocalSearch, 170	Limdo 202
RandomSearch, 172	Umda, 203 UniformCrossover, 205
reservoir_sampling	
hnco::neighborhood::MultiBitFlip, 123	update hnco::function::OnBudgetFunction, 140
Restart, 173	hnco::function::ProgressTracker, 165
Ridge, 174	hnco::function::Frogress fracker, 103
safe eval	hnco::function::StopOnTarget, 194
safe_eval hnco::function::Function, 67	update_solution
sbm_multiply	hnco::algorithm::Algorithm, 33
hnco, 19	illicoaigontilliAigontilli, oo
select	WalshExpansion, 206
hnco::algorithm::TournamentSelection, 199	WalshExpansion1, 208
set_allow_stay	WalshExpansion2, 210
hnco::algorithm::GeneticAlgorithm, 78	
hnco::algorithm::MuCommaLambdaEa, 121	
hnco::algorithm::MuPlusLambdaEa, 125	
hnco::algorithm::OnePlusOneEa, 147	
hnco::neighborhood::BernoulliProcess, 36	
set_num_iterations	
hnco::algorithm::IterativeAlgorithm, 95	
hnco::algorithm::OnePlusOneEa, 147	
set_patience	
hnco::algorithm::RandomLocalSearch, 171	
set_probability	
hnco::neighborhood::BernoulliProcess, 36	
set_reset_period	
hnco::algorithm::hea::Hea, 88	
set_selection_size	
hnco::algorithm::bm_pbil::BmPbil, 44	
hnco::algorithm::hea::Hea, 88	
set solution	
hnco::algorithm::Algorithm, 33	