HNCO

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 hnco Namespace Reference

top-level HNCO namespace

Namespaces

- namespace algorithm

  Algorithms.
- namespace app

  Classes for applications.
- namespace exception

  Exceptions.
- namespace function

  Functions defined on bit vectors.
- namespace logging

  Logging.
- namespace map

  Maps.
- namespace multiobjective

  Multiobjective optimization.
- namespace neighborhood

  Neighborhoods for local search.
- namespace random

  Random numbers.
- namespace representation

  Representations.

Classes

- class ExtendedHypercubeIterator

  Extended Hypercube iterator.
- class HypercubeIterator

  Hypercube iterator.
- class Iterator

  Iterator over bit vectors
- class StopWatch

  Stop watch.

Functions

- void ensure (bool b, const std::string message)

    Ensure that a condition is satisfied or throw a runtime exception.
- template<class A , class B >
  bool **have_same_size** (const A &a, const B &b)

    Check whether two containers have the same size.
- template<class T >
  T **square** (T x)

    Generic square function.
- double **logistic** (double x)

    Logistic function (sigmoid)
- template<typename Iter >
  std::string **join** (Iter begin, Iter end, std::string const &separator)

    Convert to string and join elements of a container (from SO)

Load from and save to boost archives

- template<typename T >
  void load_from_archive (T &object, std::string path, std::string name)

    Load from a boost archive.
- template<typename T >
  void save_to_archive (const T &object, std::string path, std::string name)

    Save to a boost archive.

Range checking

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

    Check whether an index is in a given range.
- bool is_in_range (int i, int n)

    Check whether an index is in a given range.

Intervals

- bool **is_in_interval** (double x, double a, double b)

    Check whether a double value belongs to a given interval.

Types and functions related to bit matrices

Output and input-output function parameters appear at the beginning of the parameter list.

Output and input-output bit_matrix_t parameters are passed by reference and must have the right size for the considered function.

Input object parameters are passed by const reference.

- using **bit_matrix_t** = std::vector< bit_vector_t >

    Bit matrix.
- bit_matrix_t **bm_rectangular** (int nrows, int ncols)

    Make a rectangular bit matrix.
- bit_matrix_t **bm_square** (int n)

    Make a square bit matrix.

- void bm_identity (bit_matrix_t &M)

  Set a matrix to the identity matrix.
- bit_matrix_t bm_identity (int n)

  Make an identity bit matrix.
- void bm_transpose (bit_matrix_t &N, const bit_matrix_t &M)

  Transpose a bit matrix.
- bit_matrix_t bm_transpose (const bit_matrix_t &M)

  Transpose a bit matrix.
- void **bm_display** (const bit_matrix_t &M, std::ostream &stream)

  Display bit matrix.
- bool **bm_is_valid** (const bit_matrix_t &M)

  Check whether a bit matrix is valid.
- int **bm_num_rows** (const bit_matrix_t &M)

  Number of rows.
- int **bm_num_columns** (const bit_matrix_t &M)

  Number of columns.
- bool **bm_is_square** (const bit_matrix_t &M)

  Check whether the matrix is a square matrix.
- bool **bm_is_identity** (const bit_matrix_t &M)

  Check whether the matrix is the identity matrix.
- bool **bm_is_upper_triangular** (const bit_matrix_t &M)

  Check whether the matrix is upper triangular.
- void **bm_resize** (bit_matrix_t &M, int nrows, int ncols)

  Resize a bit matrix.
- void **bm_resize** (bit_matrix_t &M, int nrows)

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

  Clear bit matrix.
- void **bm_random** (bit_matrix_t &M)

  Sample a random bit matrix.
- void **bm_swap_rows** (bit_matrix_t &M, int i, int j)

  Swap two rows.
- void bm_add_rows (bit_matrix_t &M, int dest, int src)

  Add two rows.
- void bm_add_columns (bit_matrix_t &M, int dest, int src)

  Add two columns.
- void bm_set_column (bit_matrix_t &M, int j, const bit_vector_t &bv)

  Set column.
- void bm_row_echelon_form (bit_matrix_t &A)

  Compute a row echelon form of a matrix.
- int bm_rank (const bit_matrix_t &A)

  Compute the rank of a matrix.
- bool bm_solve (bit_matrix_t &A, bit_vector_t &b)

  Solve a linear system.
- bool bm_solve_upper_triangular (bit_matrix_t &A, bit_vector_t &b)

  Solve a linear system in upper triangular form.
- bool bm_invert (bit_matrix_t &M, bit_matrix_t &N)

  Invert a bit matrix.
- void bm_multiply (bit_vector_t &y, const bit_matrix_t &M, const bit_vector_t &x)

  Multiply a bit matrix and a bit vector.

Types and functions related to bits

- using **bit_t** = std::uint8_t

  Bit.
- void bit_add (bit_t &dest, bit_t b)

  Add bits.
- void bit_add (bit_t &dest, bit_t b1, bit_t b2)

  Add bits.
- void bit_flip (bit_t &b)

  Flip a bit.
- bit_t bit_random (double p)

  Sample a random bit.

Types and functions related to bit vectors

Output and input-output function parameters appear at the beginning of the parameter list.

Output and input-output bit_vector_t parameters are passed by reference and must have the right size for the considered function.

Input bit_vector_t parameters are passed by const reference.

- using **bit_vector_t** = std::vector< bit_t >

  Bit vector.
- std::string **bv_domain** (const bit_vector_t &x)

  Display bit vector.
- 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_add (bit_vector_t &dest, const bit_vector_t &src)

  Add two bit vectors.
- void bv_add (bit_vector_t &dest, const bit_vector_t &x, const bit_vector_t &y)

  Add two bit vectors.
- void **bv_flip** (bit_vector_t &x, int i)

  Flip a single bit.

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

  Flip many bits given by a sparse bit vector.
- 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_to_vector_bool (std::vector< bool > &y, const bit_vector_t &x)

  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 small bit vector to a size_t.
- std::size_t bv_to_size_type (const bit_vector_t &x, int start, int stop)

  Convert a slice of a small bit vector to a size_t.
- void bv_from_size_type (bit_vector_t &x, std::size_t u)

  Convert a size_t to a small bit vector.
- bit_vector_t bv_from_string (const std::string &str)

  Read a bit vector from a string.
- bit_vector_t bv_from_stream (std::istream &stream)

  Read a bit vector from a stream.

Types and functions related to permutations

- using **permutation_t** = std::vector< int >

  Permutation type
- bool **perm_is_valid** (const permutation_t &permutation)

  Check that a vector represents a permutation.
- void perm_identity (permutation_t &s)

  Identity permutation.
- void **perm_shuffle** (permutation_t &s)

  Shuffle a permutation.
- void perm_random (permutation_t &s)

  Sample a random permutation.
- void **perm_display** (const permutation_t &permutation, std::ostream &stream)

  Display a permutation.

Types and functions related to sparse bit vectors

- using sparse_bit_vector_t = std::vector< int >

  Sparse bit vector.
- bool sbv_is_valid (const sparse_bit_vector_t &sbv)

  Check that a sparse bit vector is valid.
- bool sbv_is_valid (const sparse_bit_vector_t &sbv, int n)

  Check that a sparse bit vector is valid.
- void **sbv_display** (const sparse_bit_vector_t &v, std::ostream &stream)

  Display sparse bit vector.

4.1.1 Detailed Description

top-level HNCO namespace

4.1.2 Typedef Documentation

4.1.2.1 sparse_bit_vector_t

```
using sparse_bit_vector_t = std::vector<int>
```

Sparse bit vector.

A sparse bit vector is represented as an vector 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 45 of file sparse-bit-vector.hh.

4.1.3 Function Documentation

4.1.3.1 bit_add() [1/2]

```
void bit_add (
            bit_t & dest,
            bit_t b )  [inline]
```

Add bits.

Implements dest = dest xor b

Parameters

dest	Destination bit
b	Operand

Definition at line 56 of file bit-vector.hh.

4.1.3.2 bit_add() [2/2]

```
void bit_add (
            bit_t & dest,
```

```
             bit_t b1,
             bit_t b2 )   [inline]
```

Add bits.

Implements dest = b1 xor b2

Parameters

dest	Destination bit
b1	First operand
b2	Second operand

Definition at line 65 of file bit-vector.hh.

4.1.3.3 bit_flip()

```
void bit_flip (
             bit_t & b )   [inline]
```

Flip a bit.

Parameters

b	Bit to flip

Definition at line 71 of file bit-vector.hh.

4.1.3.4 bit_random()

```
bit_t bit_random (
             double p )   [inline]
```

Sample a random bit.

Parameters

p	Probability of 1

Definition at line 77 of file bit-vector.hh.

4.1.3.5 bm_add_columns()

```
void bm_add_columns (
```

```
              bit_matrix_t & M,
              int dest,
              int src )
```

Add two columns.

Equivalent to dest = dest + src.

Parameters

M	Bit matrix
dest	Destination column
src	Source column

Warning

M is modified by the function.

Definition at line 187 of file bit-matrix.cc.

4.1.3.6 bm_add_rows()

```
void bm_add_rows (
              bit_matrix_t & M,
              int dest,
              int src )
```

Add two rows.

Equivalent to dest = dest + src.

Parameters

M	Bit matrix
dest	Destination row
src	Source row

Definition at line 178 of file bit-matrix.cc.

4.1.3.7 bm_identity() [1/2]

```
void bm_identity (
              bit_matrix_t & M )
```

Set a matrix to the identity matrix.

Precondition

bm_is_square(M)

Definition at line 39 of file bit-matrix.cc.

4.1.3.8 bm_identity() [2/2]

```
bit_matrix_t bm_identity (
                int n )
```

Make an identity bit matrix.

Parameters

n	Dimension

Returns

An order n identity matrix

Definition at line 50 of file bit-matrix.cc.

4.1.3.9 bm_invert()

```
bool bm_invert (
                bit_matrix_t & M,
                bit_matrix_t & N )
```

Invert a bit matrix.

Parameters

M	Bit matrix
N	Inverse bit matrix

Precondition

bm_is_square(M)

bm_is_square(N)

bm_num_rows(M) == bm_num_rows(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 316 of file bit-matrix.cc.

4.1.3.10 bm_multiply()

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

Multiply a bit matrix and a bit vector.

Computes y = Mx.

Parameters

y	Output bit vector
M	Bit matrix
x	Bit vector

Definition at line 360 of file bit-matrix.cc.

4.1.3.11 bm_rank()

```
int bm_rank (
            const bit_matrix_t & A )
```

Compute the rank of a matrix.

Precondition

A must be in row echelon form.

Definition at line 244 of file bit-matrix.cc.

4.1.3.12 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 213 of file bit-matrix.cc.

4.1.3.13 bm_set_column()

```
void bm_set_column (
            bit_matrix_t & M,
            int j,
            const bit_vector_t & bv )
```

Set column.

Set a column to a given bit vector.

Parameters

M	Bit matrix
j	Column index
bv	Bit vector

Precondition

bm_num_rows(M) == bv.size()

Definition at line 202 of file bit-matrix.cc.

4.1.3.14 bm_solve()

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

Solve a linear system.

Solve the linear equation Ax = b.

Parameters

A	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 262 of file bit-matrix.cc.

4.1.3.15 bm_solve_upper_triangular()

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

Solve a linear system in upper triangular form.

Solve the linear equation Ax = b.

Parameters

A	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 295 of file bit-matrix.cc.

4.1.3.16 bm_transpose() [1/2]

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

Transpose a bit matrix.

Precondition

bm_num_columns(N) == bm_num_rows(M)

bm_num_rows(N) == bm_num_columns(M)

Definition at line 59 of file bit-matrix.cc.

4.1.3.17 bm_transpose() [2/2]

```
bit_matrix_t bm_transpose (
            const bit_matrix_t & M )
```

Transpose a bit matrix.

Parameters

M	Bit matrix

Returns

Transposed bit matrix

Definition at line 73 of file bit-matrix.cc.

4.1.3.18 bv_add() [1/2]

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

Add two bit vectors.

Equivalent to dest = dest + src.

Parameters

dest	Destination bit vector
src	Source bit vector

Warning

Vectors must be of the same size.

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

4.1.3.19 bv_add() [2/2]

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

Add two bit vectors.

Equivalent to dest = x + y.

Parameters

dest	Destination bit vector
x	First operand
y	Second operand

Warning

Vectors must be of the same size.

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

4.1.3.20 bv_flip()

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

Flip many bits given by a sparse bit vector.

Parameters

x	Input-output bit vector
sbv	Bits to flip

Definition at line 92 of file bit-vector.cc.

4.1.3.21 bv_from_size_type()

```
void bv_from_size_type (
            bit_vector_t & x,
            std::size_t u )
```

Convert a size_t to a small bit vector.

Parameters

x	Output bit vector
u	Unsigned integer representing a bit vector

Precondition

x.size() $<=$ 8 $*$ sizeof(std::size_t)

Warning

Depending on the size of the output bit vector, some bits might be lost. The original bit vector can be reconstructed only if it is small and the unsigned integer u is the result of bv_to_size_type.

Definition at line 202 of file bit-vector.cc.

4.1.3.22 bv_from_stream()

```
bit_vector_t bv_from_stream (
            std::istream & stream )
```

Read a bit vector from a stream.

Parameters

stream	Input stream

Returns

A bit_vector_t

Definition at line 233 of file bit-vector.cc.

4.1.3.23 bv_from_string()

```
bit_vector_t bv_from_string (
            const std::string & str )
```

Read a bit vector from a string.

Parameters

str	Input string

Returns

A bit_vector_t

Definition at line 217 of file bit-vector.cc.

4.1.3.24 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 157 of file bit-vector.cc.

4.1.3.25 bv_to_size_type() [1/2]

```
std::size_t bv_to_size_type (
            const bit_vector_t & x )
```

Convert a small bit vector to a size_t.

x[0] is the least significant bit.

Parameters

x	Input bit vector

Returns

An unsigned integer representing x

Precondition

x.size() <= 8 ∗ sizeof(std::size_t)

Definition at line 170 of file bit-vector.cc.

4.1.3.26 bv_to_size_type() [2/2]

```
std::size_t bv_to_size_type (
            const bit_vector_t & x,
            int start,
            int stop )
```

Convert a slice of a small bit vector to a size_t.

x[start] is the least significant bit.

x[stop-1] is the most significant bit.

Parameters

x	Input bit vector
start	Start bit
stop	Stop bit

Returns

An unsigned integer representing x[start], ..., x[stop-1]

Precondition

start in [0, x.size())

stop in [start+1, x.size()]

(stop - start) <= 8 ∗ sizeof(std::size_t)

Definition at line 185 of file bit-vector.cc.

4.1.3.27 bv_to_vector_bool()

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

Convert a bit vector to a bool vector.

Warning

Vectors must be of the same size.

Definition at line 144 of file bit-vector.cc.

4.1.3.28 ensure()

```
void ensure (
            bool b,
            const std::string message )  [inline]
```

Ensure that a condition is satisfied or throw a runtime exception.

Parameters

b	Boolean
message	Message to display if the boolean is false

Definition at line 39 of file util.hh.

4.1.3.29 is_in_range() [1/2]

```
bool is_in_range (
            int i,
            int a,
            int b )  [inline]
```

Check whether an index is in a given range.

Parameters

i	Index
a	Lower bound
b	Upper bound (excluded)

Returns

true if i >= a and i < b

Definition at line 58 of file util.hh.

4.1.3.30 is_in_range() [2/2]

```
bool is_in_range (
            int i,
            int n )  [inline]
```

Check whether an index is in a given range.

The lower bound is implicit and is equal to 0.

Parameters

i	Index
n	Upper bound (excluded)

Returns

true if i >= 0 and i < n

Definition at line 67 of file util.hh.

4.1.3.31 load_from_archive()

```
void load_from_archive (
            T & object,
            std::string path,
            std::string name )
```

Load from a boost archive.

Parameters

object	Object to load
path	Path of the file
name	Class name

Definition at line 44 of file serialization.hh.

4.1.3.32 perm_identity()

```
void perm_identity (
            permutation_t & s )  [inline]
```

Identity permutation.

Warning

This function does not set the size of the permutation.

Definition at line 47 of file permutation.hh.

4.1.3.33 perm_random()

```
void perm_random (
            permutation_t & s )  [inline]
```

Sample a random permutation.

Warning

This function does not set the size of the permutation.

Definition at line 60 of file permutation.hh.

4.1.3.34 save_to_archive()

```
void save_to_archive (
            const T & object,
            std::string path,
            std::string name )
```

Save to a boost archive.

Parameters

object	Object to save
path	Path of the file
name	Class name

Definition at line 64 of file serialization.hh.

4.1.3.35 sbv_is_valid() [1/2]

```
bool sbv_is_valid (
            const sparse_bit_vector_t & sbv )
```

Check that a sparse bit vector is valid.

A sparse bit vector is valid if:

- Its elements are non negative.
- Its elements are sorted in non-descending order.

Definition at line 30 of file sparse-bit-vector.cc.

4.1.3.36 sbv_is_valid() [2/2]

```
bool sbv_is_valid (
            const sparse_bit_vector_t & sbv,
            int n )
```

Check that a sparse bit vector is valid.

A sparse bit vector is valid if:

- Its elements are non negative.
- Its elements are sorted in non-descending order.
- Its elements are valid indices w.r.t. the given dimension.

Parameters

sbv	Input sparse bit vector
n	Dimension

Definition at line 41 of file sparse-bit-vector.cc.

4.2 hnco::algorithm Namespace Reference

Algorithms.

Namespaces

- namespace fast_efficient_p3

  Algorithms from the FastEfficientP3 library.
- namespace gomea

  GOMEA.
- namespace walsh_moment

  Algorithms using Walsh moments.

Classes

- class Algorithm

  Abstract search algorithm.
- class BiasedCrossover

  Biased crossover.
- class CommaSelection

  Comma selection.
- class CompactGa

  Compact genetic algorithm.
- class CompleteSearch

  Complete search.
- class Crossover

  Crossover
- class FirstAscentHillClimbing

  First ascent hill climbing.
- class GeneticAlgorithm

  Genetic algorithm.
- class Human

  Human
- class InformationTheoreticEa

  Information-theoretic evolutionary algorithm.
- class IterativeAlgorithm

  Iterative search.
- class LocalSearchAlgorithm

  Local search algorithm.
- class Mimic

Mutual information maximizing input clustering.
- 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 PlusSelection

  Plus selection.
- struct Population

  Population
- class PvAlgorithm

  Probability vector algorithm.
- class RandomLocalSearch

  Random local search.
- class RandomSearch

  Random search.
- class RandomSelection

  Random selection.
- class RandomWalk

  Random walk.
- class Restart

  Restart.
- class SelfAdjustingOnePlusOneEa

  Self-adjusting (1+1) evolutionary algorithm.
- class SimulatedAnnealing

  Simulated annealing.
- class SteepestAscentHillClimbing

  Steepest ascent hill climbing.
- class TournamentSelection

  Tournament selection.
- class TwoRateOnePlusLambdaEa

  Two-rate (1+lambda) evolutionary algorithm.
- class Umda

  Univariate marginal distribution algorithm.
- class UniformCrossover

  Uniform crossover.
- class UniformSelection

  Uniform selection.

Typedefs

- using **solution_t** = std::pair< bit_vector_t, double >

  Type of a solution.

Functions

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

  Check for symmetric matrix.
- template<class T >
  bool **matrix_is_strictly_lower_triangular** (const std::vector< std::vector< T > > &A)

  Check for strictly lower triangular matrix.
- template<class T >
  bool **matrix_has_diagonal** (const std::vector< std::vector< T > > &A, T x)

  Check for diagonal elements.
- template<class T >
  bool **matrix_has_range** (const std::vector< std::vector< T > > &A, T inf, T sup)

  Check for element range.
- template<class T >
  bool **matrix_has_dominant_diagonal** (const std::vector< std::vector< T > > &A)

  Check for element range.

Type and functions related to probability vectors

Output and input-output function parameters appear at the beginning of the parameter list.

Output and input-output pv_t parameters are passed by reference and must have the right size for the considered function.

Input object parameters are passed by const reference.

- using **pv_t** = std::vector< double >

  Probability vector type.
- double **pv_entropy** (const pv_t &pv)

  Entropy of a probability vector.
- void pv_sample (bit_vector_t &x, const pv_t &pv)

  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 into a probability vector.
- void pv_average (pv_t &pv, int count)

  Average.
- template<class T >
  void pv_update (pv_t &pv, double rate, const T &x)

  Update a probability vector.
- void pv_update (pv_t &pv, double rate, const pv_t &x, const pv_t &y)

  Update a probability vector.
- void pv_bound (pv_t &pv, double lower_bound, double upper_bound)

  Bound the elements of a probability vector.

4.2.1 Detailed Description

Algorithms.

4.2.2 Function Documentation

4.2.2.1 pv_add()

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

Accumulate a bit vector into a probability vector.

Equivalent to pv += x

Parameters

pv	Probability vector
x	Bit vector

Definition at line 58 of file probability-vector.cc.

4.2.2.2 pv_average()

```
void pv_average (
            pv_t & pv,
            int count )
```

Average.

Equivalent to pv = pv / count.

Parameters

pv	Probability vector
count	Number of accumulated bit vectors

Definition at line 67 of file probability-vector.cc.

4.2.2.3 pv_bound()

```
void pv_bound (
```

```
            pv_t & pv,
            double lower_bound,
            double upper_bound )
```

Bound the elements of a probability vector.

Parameters

pv	Probability vector
lower_bound	Lower bound
upper_bound	Upper bound

Definition at line 82 of file probability-vector.cc.

4.2.2.4 pv_init()

```
void pv_init (
            pv_t & pv )  [inline]
```

Initialize.

All the elements of the probability vector are set to 0.

Parameters

pv	Probability vector

Definition at line 74 of file probability-vector.hh.

4.2.2.5 pv_sample()

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

Sample a bit vector.

Parameters

x	Sampled bit vector
pv	Probability vector

Definition at line 46 of file probability-vector.cc.

4.2.2.6 pv_uniform()

```
void pv_uniform (
            pv_t & pv )  [inline]
```

Probability vector of the uniform distribution.

All the elements of the probability vector are set to 1/2.

Parameters

pv	Probability vector

Definition at line 66 of file probability-vector.hh.

4.2.2.7 pv_update() [1/2]

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

Update a probability vector.

Equivalent to pv += rate(x - y)

Parameters

pv	Probability vector
rate	Rate
x	Attractor probability vector
y	Repulsor probability vector

Definition at line 73 of file probability-vector.cc.

4.2.2.8 pv_update() [2/2]

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

Update a probability vector.

Equivalent to pv += rate ∗ (x - pv)

Parameters

pv	Probability vector
rate	Rate
x	Attractor bit vector

Definition at line 103 of file probability-vector.hh.

4.3 hnco::algorithm::fast_efficient_p3 Namespace Reference

Algorithms from the FastEfficientP3 library.

Classes

- class Hboa

  Hierarchical Bayesian Optimization Algorithm.
- class HncoEvaluator

  Evaluator for HNCO functions.
- struct Implementation

  Implementation
- class Ltga

  Linkage Tree Genetic Algorithm.
- class ParameterLessPopulationPyramid

  Parameter-less Population Pyramid.

4.3.1 Detailed Description

Algorithms from the FastEfficientP3 library.

4.4 hnco::algorithm::gomea Namespace Reference

GOMEA.

Classes

- class Gomea

  GOMEA.
- class HncoFitness

  Fitness for HNCO functions.

4.4.1 Detailed Description

GOMEA.

4.5 hnco::algorithm::walsh_moment Namespace Reference

Algorithms using Walsh moments.

Classes

- class BmPbil

  Boltzmann machine PBIL.
- class Hea

  Herding evolutionary algorithm.
- struct LowerTriangularWalshMoment2

  Lower triangular Walsh moment.
- class LowerTriangularWalshMoment2GibbsSampler

  Gibbs sampler with lower triangular Walsh moments.
- class LowerTriangularWalshMoment2Herding

  Herding with lower triangular Walsh moment.
- struct SymmetricWalshMoment2

  Symmetric Walsh moment.
- class SymmetricWalshMoment2GibbsSampler

  Gibbs sampler with symmetric Walsh moments.
- class SymmetricWalshMoment2Herding

  Herding with symmetric Walsh moment.

4.5.1 Detailed Description

Algorithms using Walsh moments.

4.6 hnco::app Namespace Reference

Classes for applications.

Classes

- class AlgorithmFactory

  Algorithm factory.
- class CommandLineAlgorithmFactory

  Command line algorithm factory.
- class CommandLineApplication

  Command line application.
- class CommandLineFunctionFactory

  Command line function factory.
- class DecoratedFunctionFactory

  Decorated function factory.
- class FfgenOptions

  Command line options for ffgen.
- class FunctionFactory

  Function factory.
- class HncoOptions

  Command line options for hnco.
- class MapgenOptions

  Command line options for mapgen.

Typedefs

- using **IntRep** = representation::DyadicIntegerRepresentation< int >
  Int representation.
- using **LongRep** = representation::DyadicIntegerRepresentation< long >
  Long representation.
- using **DoubleRep** = representation::DyadicFloatRepresentation< double >
  Double representation.

Functions

- std::ostream & **operator**<< (std::ostream &stream, const HncoOptions &options)
  Print a header containing the parameter values.
- std::string **read_file_content** (std::string path)
  Read file content.
- std::vector< std::string > **split_string** (std::string str, std::string delimiter)
  Split string.
- template<typename Options >
  env_t parse_representations (std::string expression, const Options &options)
  Parse representations.
- template<typename Options , typename Adapter >
  Adapter ∗ **make_multivariate_function_adapter** (const Options &options)
  Make a multivariate function adapter.
- template<typename Options , typename Adapter >
  Adapter ∗ **make_multivariate_function_adapter_complex** (const Options &options)
  Make a multivariate function adapter over complex domain.
- template<typename Options , typename Adapter >
  Adapter ∗ **make_multivariate_function_adapter_mixed** (const Options &options)
  Make a mixed-integer multivariate function adapter.
- std::ostream & **operator**<< (std::ostream &stream, const FfgenOptions &options)
  Print a header containing the parameter values.
- std::ostream & **operator**<< (std::ostream &stream, const MapgenOptions &options)
  Print a header containing the parameter values.

4.6.1 Detailed Description

Classes for applications.

4.6.2 Function Documentation

4.6.2.1 parse_representations()

```
env_t parse_representations (
            std::string expression,
            const Options & options )
```

Parse representations.

Parameters

expression	Expression to parse
options	Options

Syntax:

representations = declaration [; declaration]∗

declaration = name : representation

representation =

- int(a, b) where a, b are int
- long(a, b) where a, b are long
- double(a, b, precision = e) where a, b, e are double
- double(a, b, size = n) where a, b are double, and n is int

Example:

"x: double(0, 1); y: double(0, 1, precision = 1e-3); z: double(0, 1, size = 8); u: int(-100, 100); v: long(1, 10000)"

Definition at line 242 of file make-multivariate-function-adapter.hh.

4.7 hnco::exception Namespace Reference

Exceptions.

Classes

- class LastEvaluation

    Last evaluation.
- class TargetReached

    Target reached.

4.7.1 Detailed Description

Exceptions.

4.8 hnco::function Namespace Reference

Functions defined on bit vectors.

Namespaces

- namespace controller

  Controllers.
- namespace modifier

  Modifiers.

Classes

- class AbstractMaxSat

  Abstract class for MaxSat-like functions.
- struct ComplexToDouble

  Convert a complex to a double.
- class DeceptiveJump

  Deceptive jump.
- class Decorator

  Function decorator
- class EqualProducts

  Equal products.
- class Factorization

  Factorization.
- class FourPeaks

  Four Peaks.
- class Function

  Function
- 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 MaxNae3Sat

  Max not-all-equal 3SAT.
- class MaxSat

  MAX-SAT.
- class MixedIntegerMultivariateFunctionAdapter

  Mixed-integer multivariate function adapter.
- class MultivariateFunctionAdapter

  Multivariate function adapter.
- class NearestNeighborIsingModel1

  Nearest neighbor Ising model in one dimension.
- class NearestNeighborIsingModel2

Nearest neighbor Ising model in two dimensions.
- class Needle

  Needle in a haystack.
- class NkLandscape

  NK landscape.
- class OneMax

  OneMax.
- class ParsedMultivariateFunction

  Parsed multivariate function.
- class Partition

  Partition.
- class PermutationFunctionAdapter

  Permutation function adapter.
- class Plateau

  Plateau.
- class PythonFunction

  Python function.
- class Qubo

  Quadratic unconstrained binary optimization.
- class Ridge

  Ridge.
- struct ScalarToDouble

  Convert a scalar to a double.
- class SinusSummationCancellation

  Summation cancellation with sinus.
- class SixPeaks

  Six Peaks.
- class Sudoku

  Sudoku
- class SummationCancellation

  Summation cancellation.
- class Trap

  Trap.
- class Tsp

  Traveling salesman problem.
- class UniversalFunction

  Universal function.
- class UniversalFunctionAdapter

  Universal function adapter.
- class WalshExpansion

  Walsh expansion.
- class WalshExpansion1

  Walsh expansion of degree 1.
- class WalshExpansion2

  Walsh expansion of degree 2.
- struct WalshTerm

  Walsh transform term.

Functions

- void compute_walsh_transform (function::Function $*$function, std::vector$<$ function::WalshTerm $>$ &terms)

  Compute the Walsh transform of the function.
- void compute_fast_walsh_transform (function::Function $*$function, std::vector$<$ function::WalshTerm $>$ &terms)

  Compute the Walsh transform of the function using a fast Walsh transform.
- bool **bv_is_locally_maximal** (const bit_vector_t &bv, Function &fn, neighborhood::NeighborhoodIterator &it)

  Check whether a bit vector is locally maximal.
- bool **bv_is_globally_maximal** (const bit_vector_t &bv, Function &fn)

  Check whether a bit vector is globally maximal.

4.8.1 Detailed Description

Functions defined on bit vectors.

4.8.2 Function Documentation

4.8.2.1 compute_fast_walsh_transform()

```
void compute_fast_walsh_transform (
            function::Function * function,
            std::vector< function::WalshTerm > & terms )
```

Compute the Walsh transform of the function using a fast Walsh transform.

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. It is also helpful to achieve exact computations in the case of functions taking only integer values.

Parameters

function	Function the Walsh transform of which to compute
terms	Vector of non zero terms of the Walsh transform

Warning

The time complexity is exponential in the dimension n. It requires 2^n function evaluations and $n2^n$ additions, which is faster than compute_walsh_transform.

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 77 of file function.cc.

4.8.2.2 compute_walsh_transform()

```
void compute_walsh_transform (
            function::Function * function,
            std::vector< function::WalshTerm > & terms )
```

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. It is also helpful to achieve exact computations in the case of functions taking only integer values.

Parameters

function	Function the Walsh transform of which to compute
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.

4.9 hnco::function::controller Namespace Reference

Controllers.

Classes

- class Cache

  Cache.
- class CallCounter

  Call counter.
- class Controller

  Function controller.
- class OnBudgetFunction

  Function with a limited number of evaluations.
- class ProgressTracker

  Progress tracker.
- class StopOnMaximum

  Stop on maximum.
- class StopOnTarget

  Stop on target.

Functions

- std::ostream & **operator**<< (std::ostream &stream, const ProgressTracker::Event &event)

    Insert formatted output.

4.9.1 Detailed Description

Controllers.

4.10 hnco::function::modifier Namespace Reference

Modifiers.

Classes

- class AdditiveGaussianNoise

    Additive Gaussian Noise.
- class FunctionMapComposition

    Composition of a function and a map.
- class Modifier

    Function modifier.
- class Negation

    Negation.
- class ParsedModifier

    Parsed modifier.
- class PriorNoise

    Prior noise.

4.10.1 Detailed Description

Modifiers.

4.11 hnco::logging Namespace Reference

Logging.

Classes

- class LogContext

    Log context.
- class Logger

    Logger.
- class ProgressTrackerContext

    Log context for ProgressTracker.

4.11.1 Detailed Description

Logging.

4.12 hnco::map Namespace Reference

Maps.

Classes

- class AffineMap

    Affine map.
- class Injection

    Injection.
- class LinearMap

    Linear map.
- class Map

    Map
- class MapComposition

    Map composition.
- class Permutation

    Permutation.
- class Projection

    Projection.
- class Translation

    Translation.
- struct Transvection

    Transvection.
- class TsAffineMap

    Transvection sequence affine map.

Types and functions related to transvections

Output and input-output function parameters appear at the beginning of the parameter list.

Output and input-output transvection_sequence_t parameters are passed by reference.

Input object parameters are passed by const reference.

- using transvection_sequence_t = std::vector< Transvection >

    Transvection sequence.
- bool **transvections_commute** (const Transvection &a, const Transvection &b)

    Check whether two transvections commute.
- bool **transvections_are_disjoint** (const Transvection &a, const Transvection &b)

    Check whether two transvections are disjoint.
- bool ts_is_valid (const transvection_sequence_t &ts)

    Check validity.

- bool ts_is_valid (const transvection_sequence_t &ts, int n)
  Check validity.
- void **ts_display** (const transvection_sequence_t &ts, std::ostream &stream)
  Display a transvection sequence.
- void ts_random (transvection_sequence_t &ts, int n, int t)
  Sample a random transvection sequence.
- void ts_random_commuting (transvection_sequence_t &ts, int n, int t)
  Sample a random sequence of commuting transvections.
- void ts_random_unique_source (transvection_sequence_t &ts, int n, int t)
  Sample a random sequence of transvections with unique source.
- void ts_random_unique_destination (transvection_sequence_t &ts, int n, int t)
  Sample a random sequence of transvections with unique destination.
- void ts_random_disjoint (transvection_sequence_t &ts, int n, int t)
  Sample a random sequence of disjoint transvections.
- void ts_random_non_commuting (transvection_sequence_t &ts, int n, int t)
  Sample a random sequence of non commuting transvections.
- void ts_multiply (bit_vector_t &x, const transvection_sequence_t &ts)
  Multiply a vector by a transvection sequence from the left.
- void ts_multiply (bit_matrix_t &M, const transvection_sequence_t &ts)
  Multiply a matrix by a transvection sequence from the left.

4.12.1 Detailed Description

Maps.

4.12.2 Typedef Documentation

4.12.2.1 transvection_sequence_t

```
using transvection_sequence_t = std::vector<Transvection>
```

Transvection sequence.

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

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

Finite transvection sequences can then represent all invertible bit matrices.

Definition at line 166 of file transvection.hh.

4.12.3 Function Documentation

4.12.3.1 ts_is_valid() [1/2]

```
bool ts_is_valid (
            const transvection_sequence_t & ts )
```

Check validity.

Parameters

ts	Transvection sequence

Definition at line 150 of file transvection.cc.

4.12.3.2 ts_is_valid() [2/2]

```
bool ts_is_valid (
             const transvection_sequence_t & ts,
             int n )
```

Check validity.

Parameters

ts	Transvection sequence
n	Dimension

Definition at line 156 of file transvection.cc.

4.12.3.3 ts_multiply() [1/2]

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

Multiply a matrix by a transvection sequence from the left.

Parameters

ts	Transvection sequence
M	Bit matrix

Precondition

ts_is_valid(ts)

ts_is_valid(ts, bm_num_rows(M))

Warning

This function modifies the given bit vector.

Definition at line 366 of file transvection.cc.

4.12.3.4 ts_multiply() [2/2]

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

Multiply a vector by a transvection sequence from the left.

Parameters

ts	Transvection sequence
x	Bit vector

Precondition

ts_is_valid(ts)

ts_is_valid(ts, x.size())

Warning

This function modifies the given bit vector.

Definition at line 356 of file transvection.cc.

4.12.3.5 ts_random()

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

Sample a random transvection sequence.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n > 1

t >= 0

Definition at line 172 of file transvection.cc.

4.12.3.6 ts_random_commuting()

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

Sample a random sequence of commuting transvections.

This function ensures that all transvections in the sequence commute.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n $>$ 1

t $>$= 0

Warning

If t $>$ floor(n / 2) then t is set to floor(n / 2).

If t = floor(n / 2) then the space and time complexity of ts_random_commuting is quadratic in the dimension n.

Definition at line 183 of file transvection.cc.

4.12.3.7 ts_random_disjoint()

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

Sample a random sequence of disjoint transvections.

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

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

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n > 1

t >= 0

Definition at line 311 of file transvection.cc.

4.12.3.8 ts_random_non_commuting()

```
void ts_random_non_commuting (
            transvection_sequence_t & ts,
            int n,
            int t )
```

Sample a random sequence of non commuting transvections.

This function ensures that two consecutive transvections do not commute.

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n > 1

t >= 0

Definition at line 341 of file transvection.cc.

4.12.3.9 ts_random_unique_destination()

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

Sample a random sequence of transvections with unique destination.

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

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n > 1

t >= 0

Definition at line 278 of file transvection.cc.

4.12.3.10 ts_random_unique_source()

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

Sample a random sequence of transvections with unique source.

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

Parameters

ts	Transvection sequence
n	Dimension
t	Length of the sequence

Precondition

n > 1

t >= 0

Definition at line 245 of file transvection.cc.

4.13 hnco::multiobjective Namespace Reference

Multiobjective optimization.

Namespaces

- namespace algorithm

  Multiobjective Algorithms.
- namespace app

  Classes for applications.
- namespace function

  Functions defined on bit vectors.

4.13.1 Detailed Description

Multiobjective optimization.

4.14 hnco::multiobjective::algorithm Namespace Reference

Multiobjective Algorithms.

Classes

- class Algorithm
  Abstract multiobjective search algorithm.
- struct FrontDistancePair
  Front-distance pair.
- class IterativeAlgorithm
  Iterative algorithm.
- class Nsga2
  NSGA-II.
- class Nsga2ParetoFrontComputation
  Pareto front computation from the NSGA-II paper.
- struct Population
  Population
- class TournamentSelection
  Tournament selection.

Functions

- bool operator< (const FrontDistancePair &a, const FrontDistancePair &b)
  Comparison operator for front-distance pairs.

4.14.1 Detailed Description

Multiobjective Algorithms.

4.14.2 Function Documentation

4.14.2.1 operator<()

```
bool operator< (
            const FrontDistancePair & a,
            const FrontDistancePair & b )  [inline]
```

Comparison operator for front-distance pairs.

Favors individuals with smaller Pareto front then greater crowding distance.

Definition at line 61 of file nsga2.hh.

4.15 hnco::multiobjective::app Namespace Reference

Classes for applications.

Classes

- class AlgorithmFactory
  Algorithm factory.
- class CommandLineAlgorithmFactory
  Command line algorithm factory.
- class CommandLineApplication
  Command line application.
- class CommandLineFunctionFactory
  Command line function factory.
- class FunctionFactory
  Function factory.
- class HncoOptions
  Command line options for hnco-mo.

Functions

- std::ostream & **operator**<< (std::ostream &stream, const HncoOptions &options)
  Print a header containing the parameter values.

4.15.1 Detailed Description

Classes for applications.

4.16 hnco::multiobjective::function Namespace Reference

Functions defined on bit vectors.

Classes

- class Function
  Function
- class MixedIntegerMultivariateFunctionAdapter
  Mixed-integer multivariate function adapter.
- class MultivariateFunctionAdapter
  Multivariate function adapter.
- class ParsedMultivariateFunction
  Parsed multivariate function.
- class PythonFunction
  Python function.
- class UniversalFunction
  Universal function.
- class UniversalFunctionAdapter
  Universal function adapter.

Typedefs

- using value_t = std::vector< double >
  Value type.

Functions

- bool dominates (const value_t &a, const value_t &b)
  Domination relation.
- void **value_display** (const value_t &a, std::ostream &stream)
  Display a value.

4.16.1 Detailed Description

Functions defined on bit vectors.

4.16.2 Typedef Documentation

4.16.2.1 value_t

```
using value_t = std::vector<double>
```

Value type.

A value type is the type of the output of a Function in the context of multiobjective optimization.

Definition at line 42 of file value.hh.

4.16.3 Function Documentation

4.16.3.1 dominates()

```
bool dominates (
            const value_t & a,
            const value_t & b )  [inline]
```

Domination relation.

Parameters

a	First value
b	Second value

Returns

true if a dominates b with respect to minimization

Definition at line 51 of file value.hh.

4.17 hnco::neighborhood Namespace Reference

Neighborhoods for local search.

Classes

- 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.
- class StandardBitMutation
  Standard bit mutation.

4.17.1 Detailed Description

Neighborhoods for local search.

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

4.18 hnco::random Namespace Reference

Random numbers.

Classes

- struct Generator
  Random number generator.

4.18.1 Detailed Description

Random numbers.

4.19 hnco::representation Namespace Reference

Representations.

Classes

- class ComplexRepresentation

  Complex representation.
- class DyadicFloatRepresentation

  Dyadic float representation.
- class DyadicIntegerRepresentation

  Dyadic integer representation.
- class IntegerCategoricalRepresentation

  Integer categorical representation.
- class LinearCategoricalRepresentation

  Linear categorical representation.
- class PermutationRepresentation

  Permutation representation.

Functions

- template<class T >
  bool difference_is_safe (T a, T b)

  Check whether the difference is safe.

4.19.1 Detailed Description

Representations.

4.19.2 Function Documentation

4.19.2.1 difference_is_safe()

```
bool difference_is_safe (
            T a,
            T b )
```

Check whether the difference is safe.

The template parameter T must be an integral type such as int or long.

The difference b - a is safe if it can be represented by the type of a and b, i.e. there is no overflow.

Parameters

a	Smallest value
b	Greatest value

Precondition

a $<$ b

Definition at line 51 of file integer.hh.

Chapter 5

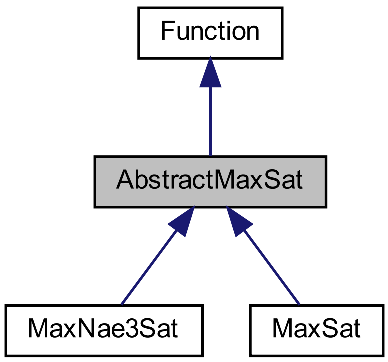
Class Documentation

5.1 AbstractMaxSat Class Reference

Abstract class for MaxSat-like functions.

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

Inheritance diagram for AbstractMaxSat:



Public Member Functions

- **AbstractMaxSat** ()

  Default constructor.
- int **get_bv_size** () const override

  Get bit vector size.
- void **display** (std::ostream &stream) const override

  Display the expression.

Load and save instance

- void load (std::string path)

  Load instance.
- void save (std::string path) const

  Save instance.

Protected Member Functions

- void load_ (std::istream &stream)

  Load an instance.
- void save_ (std::ostream &stream) const

  Save an instance.

Protected Attributes

- std::vector< std::vector< int > > _expression

  Expression.
- int **_num_variables**

  Number of variables.

5.1.1 Detailed Description

Abstract class for MaxSat-like functions.

Definition at line 36 of file max-sat.hh.

5.1.2 Member Function Documentation

5.1.2.1 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 87 of file max-sat.hh.

5.1.2.2 load_()

```
void load_ (
            std::istream & stream )  [protected]
```

Load an instance.

Parameters

stream	Input stream

Exceptions

std::runtime_error	

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

5.1.2.3 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 99 of file max-sat.hh.

5.1.2.4 save_()

```
void save_ (
            std::ostream & stream ) const  [protected]
```

Save an instance.

Parameters

stream	Outputstream

Definition at line 153 of file max-sat.cc.

5.1.3 Member Data Documentation

5.1.3.1 _expression

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

Expression.

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

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

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

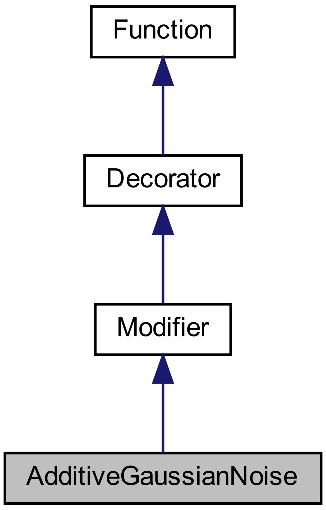
- lib/hnco/functions/collection/max-sat.hh
- lib/hnco/functions/collection/max-sat.cc

5.2 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

`#include <hnco/functions/modifiers/modifier.hh>`

Inheritance diagram for AdditiveGaussianNoise:



Public Member Functions

- **AdditiveGaussianNoise** (Function ∗function, double stddev)
  Constructor.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.

Private Attributes

- std::normal_distribution< double > **_dist**
  Normal distribution.

Additional Inherited Members

5.2.1 Detailed Description

Additive Gaussian Noise.

Definition at line 170 of file modifier.hh.

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

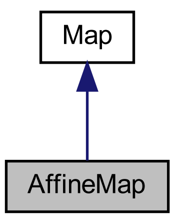
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.3 AffineMap Class Reference

Affine map.

`#include <hnco/maps/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) override

    Map
- int **get_input_size** () const override

    Get input size.
- int **get_output_size** () const override

    Get output size.
- bool is_surjective () const override

    Check for surjective map.
- void **display** (std::ostream &stream) const override

    Display.

Load and save map

- void load (std::string path)

    Load map.
- void save (std::string path) const

    Save 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

5.3.1 Detailed Description

Affine map.

An affine map f from F_2^m to F_2^n is defined by $f(x) = Ax+b$, where A is an n x m bit matrix and b is an n-dimensional bit vector.

Definition at line 331 of file map.hh.

5.3.2 Member Function Documentation

5.3.2.1 is_surjective()

```
bool is_surjective ( ) const  [override], [virtual]
```

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 149 of file map.cc.

5.3.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 405 of file map.hh.

5.3.2.3 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

std::runtime_error	

Definition at line 114 of file map.cc.

5.3.2.4 save()

```
void save (
            std::string path ) const  [inline]
```

Save map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 412 of file map.hh.

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

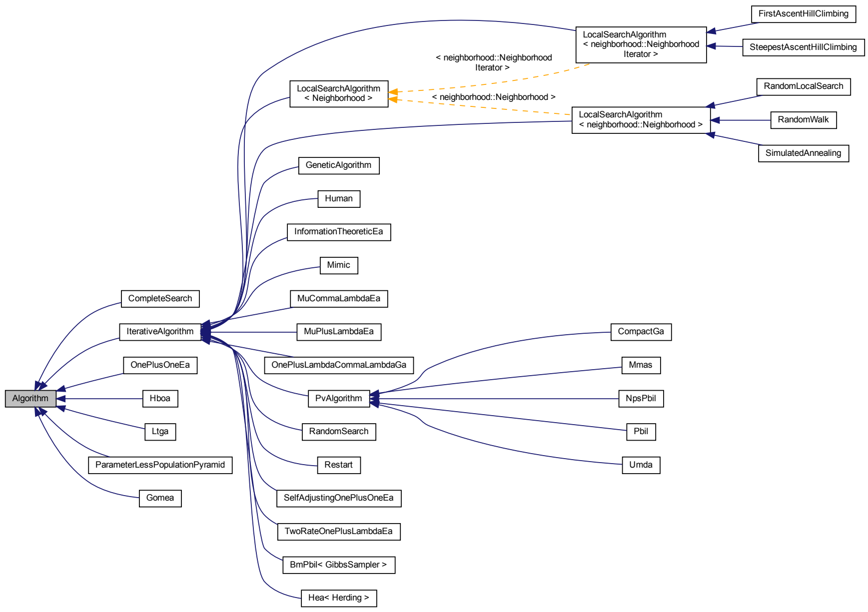
- lib/hnco/maps/map.hh
- lib/hnco/maps/map.cc

5.4 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.
- void **set_log_context** (logging::LogContext ∗log_context)

  Set the log context.

Optimization

- virtual void **maximize** (const std::vector< function::Function ∗ > &functions)=0

  Maximize.
- virtual void finalize ()

  Finalize.
- const solution_t & **get_solution** ()

  Get the solution.

Protected Member Functions

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

  Set functions.
- int **get_bv_size** ()

  Get bit vector size.

Managing solution

- void **random_solution** ()

Random solution.
- void **set_solution** (const bit_vector_t &bv, double value)
  Set solution.
- void set_solution (const bit_vector_t &bv)
  Set solution.
- void **update_solution** (const bit_vector_t &bv, double value)
  Update solution (strict)
- void **update_solution** (const solution_t &s)
  Update solution (strict)
- void update_solution (const bit_vector_t &bv)
  Update solution (strict).

Protected Attributes

- std::vector< function::Function * > _functions
  Functions.
- function::Function * **_function**
  Function.
- solution_t **_solution**
  Solution.

Parameters

- logging::LogContext * **_log_context** = nullptr
  Log context.

5.4.1 Detailed Description

Abstract search algorithm.

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

Definition at line 46 of file algorithm.hh.

5.4.2 Member Function Documentation

5.4.2.1 finalize()

```
virtual void finalize ( ) [inline], [virtual]
```

Finalize.

Does nothing.

It is usually overridden by algorithms which do not keep _solution up-to-date. In case _function throws a Last↵ Evaluation exception, the algorithm might leave _solution in an undefined state. This can be fixed in this member function.

Reimplemented in Hboa, Ltga, ParameterLessPopulationPyramid, Gomea, OnePlusOneEa, SelfAdjustingOnePlusOneEa, and RandomLocalSearch.

Definition at line 143 of file algorithm.hh.

5.4.2.2 set_solution()

```
void set_solution (
            const bit_vector_t & bv )  [protected]
```

Set solution.

Warning

Evaluates the function once.

Definition at line 45 of file algorithm.cc.

5.4.2.3 update_solution()

```
void update_solution (
            const bit_vector_t & bv )  [protected]
```

Update solution (strict).

Warning

Evaluates the function once.

Definition at line 69 of file algorithm.cc.

5.4.3 Member Data Documentation

5.4.3.1 _functions

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

Functions.

Each thread has its own function.

Definition at line 54 of file algorithm.hh.

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

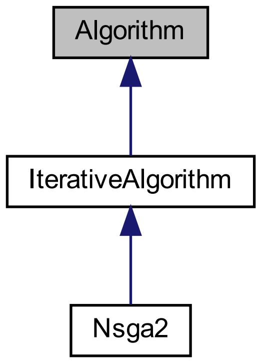
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

5.5 Algorithm Class Reference

Abstract multiobjective search algorithm.

`#include <hnco/multiobjective/algorithms/algorithm.hh>`

Inheritance diagram for Algorithm:



Public Types

- using **Function** = hnco::multiobjective::function::Function

  Function type.

Public Member Functions

- Algorithm (int n, int num_objectives)

  Constructor.
- virtual ∼**Algorithm** ()

  Destructor.
- void **set_log_context** (logging::LogContext ∗log_context)

  Set the log context.

Optimization

- virtual void **minimize** (const std::vector< Function ∗ > &functions)=0

  Minimize.
- virtual const Population & **get_solutions** ()=0

  Get solutions.

Protected Member Functions

- void **set_functions** (const std::vector< Function ∗ > &functions)

  Set functions.

Protected Attributes

- std::vector< Function * > _functions

    Functions.
- Function * **_function**

    Function.

Parameters

- logging::LogContext * **_log_context** = nullptr

    Log context.

5.5.1 Detailed Description

Abstract multiobjective search algorithm.

All algorithms minimize some given function.

Definition at line 43 of file algorithm.hh.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Algorithm()

```
Algorithm (
              int n,
              int num_objectives )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors
num_objectives	Number of objectives

Definition at line 85 of file algorithm.hh.

5.5.3 Member Data Documentation

5.5.3.1 _functions

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

Functions.

Each thread has its own function.

Definition at line 56 of file algorithm.hh.

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

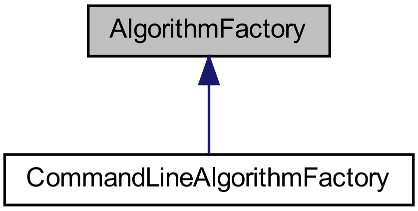
- lib/hnco/multiobjective/algorithms/algorithm.hh

5.6 AlgorithmFactory Class Reference

Algorithm factory.

```
#include <hnco/app/algorithm-factory.hh>
```

Inheritance diagram for AlgorithmFactory:



Public Member Functions

- virtual hnco::algorithm::Algorithm ∗ make (int bv_size)=0
  Make an algorithm.

5.6.1 Detailed Description

Algorithm factory.

Definition at line 32 of file algorithm-factory.hh.

5.6.2 Member Function Documentation

5.6.2.1 make()

```
virtual hnco::algorithm::Algorithm * make (
        int bv_size ) [pure virtual]
```

Make an algorithm.

Parameters

bv_size	Bit vector size

Implemented in CommandLineAlgorithmFactory.

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

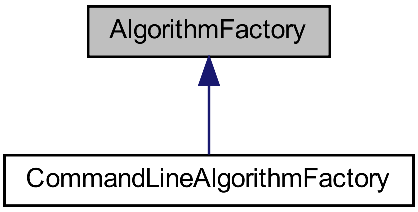
- lib/hnco/app/algorithm-factory.hh

5.7 AlgorithmFactory Class Reference

Algorithm factory.

```
#include <hnco/multiobjective/app/algorithm-factory.hh>
```

Inheritance diagram for AlgorithmFactory:



Public Member Functions

- virtual hnco::multiobjective::algorithm::Algorithm ∗ make (int bv_size, int num_objectives)=0

  Make an algorithm.

5.7.1 Detailed Description

Algorithm factory.

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

5.7.2 Member Function Documentation

5.7.2.1 make()

```
virtual hnco::multiobjective::algorithm::Algorithm * make (
            int bv_size,
            int num_objectives )  [pure virtual]
```

Make an algorithm.

Parameters

bv_size	Bit vector size

Implemented in CommandLineAlgorithmFactory.

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

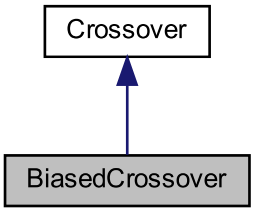
- lib/hnco/multiobjective/app/algorithm-factory.hh

5.8 BiasedCrossover Class Reference

Biased crossover.

`#include <hnco/algorithms/evolutionary-algorithms/crossover.hh>`

Inheritance diagram for BiasedCrossover:



Public Member Functions

- **BiasedCrossover** ()
  Constructor.
- void recombine (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)
  Recombine.
- void **set_bias** (double b)
  Set bias.

Private Attributes

- std::bernoulli_distribution **_bernoulli_dist**
  Bernoulli distribution.

5.8.1 Detailed Description

Biased crossover.

Definition at line 75 of file crossover.hh.

5.8.2 Member Function Documentation

5.8.2.1 recombine()

```
void recombine (
            const bit_vector_t & parent1,
            const bit_vector_t & parent2,
            bit_vector_t & offspring )  [virtual]
```

Recombine.

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:

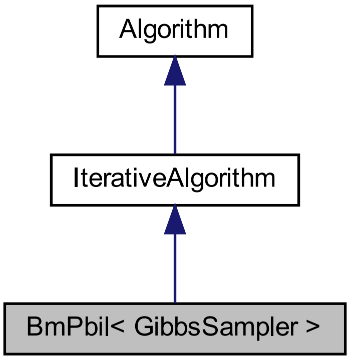
- lib/hnco/algorithms/evolutionary-algorithms/crossover.hh
- lib/hnco/algorithms/evolutionary-algorithms/crossover.cc

5.9 BmPbil< GibbsSampler > Class Template Reference

Boltzmann machine PBIL.

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

Inheritance diagram for BmPbil< GibbsSampler >:



Public Types

- enum class SamplingMode { asynchronous , asynchronous_full_scan , synchronous }

  Markov chain sampling mode.
- enum class ResetMode { no_reset , iteration , bit_vector }

  Markov chain reset mode.

Public Member Functions

- **BmPbil** (int n, int population_size)

  Constructor.

Setters for parameters

- void set_selection_size (int x)

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

  Set the learning rate.
- void **set_num_gs_steps** (int x)

  Set the number of gibbs sampler steps.
- void **set_num_gs_cycles** (int x)

  Set the number of gibbs sampler cycles.
- void **set_negative_positive_selection** (bool x)

  Set negative and positive selection.
- void **set_sampling_mode** (SamplingMode mode)

  Set the sampling mode.
- void **set_reset_mode** (ResetMode mode)

  Set the reset mode.

Setters for logging

- void **set_log_norm_infinite** (bool x)

  Log infinite norm of the model parameters.
- void **set_log_norm_1** (bool x)

  Log 1-norm of the model parameters.

Protected Member Functions

- void **set_something_to_log** ()

  Set flag for something to 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.

Loop

- void **init** () override

  Initialize.
- void **iterate** () override

  Single iteration.
- void **log** () override

  Log.

Protected Attributes

- Population **_population**

  Population.
- GibbsSampler::Moment **_model_parameters**

  Model parameters.
- GibbsSampler **_gibbs_sampler**

  Model.
- GibbsSampler::Moment **_walsh_moment_all**

  Parameters averaged over all individuals.
- GibbsSampler::Moment **_walsh_moment_best**

  Parameters averaged over selected individuals.
- GibbsSampler::Moment **_walsh_moment_worst**

  Parameters averaged over negatively selected individuals.
- std::uniform_int_distribution< int > **_choose_bit**

  Uniform distribution on bit_vector_t components.
- permutation_t **_permutation**

  Permutation.

Parameters

- int **_selection_size** = 1

  Selection size (number of selected individuals in the population)
- double **_learning_rate** = 1e-3

  Learning rate.
- int **_num_gs_steps** = 100

  Number of gibbs sampler steps.
- int **_num_gs_cycles** = 1

  Number of gibbs sampler cycles.
- bool **_negative_positive_selection** = false

Negative and positive selection.
- SamplingMode **_sampling_mode** = SamplingMode::asynchronous
  Sampling mode.
- ResetMode **_reset_mode** = ResetMode::no_reset
  Reset mode.

Logging

- bool **_log_norm_infinite** = false
  Log infinite norm of the model parameters.
- bool **_log_norm_1** = false
  Log 1-norm of the model parameters.

5.9.1 Detailed Description

template<class GibbsSampler>
class hnco::algorithm::walsh_moment::BmPbil< GibbsSampler >

Boltzmann machine PBIL.

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

Reference:

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

Definition at line 50 of file bm-pbil.hh.

5.9.2 Member Enumeration Documentation

5.9.2.1 ResetMode

```
enum class ResetMode  [strong]
```

Markov chain reset mode.

Enumerator

no_reset	No reset.
iteration	Reset the Markov chain at the beginning of each iteration.
bit_vector	Reset the Markov chain before sampling each bit vector.

Definition at line 84 of file bm-pbil.hh.

5.9.2.2 SamplingMode

```
enum class SamplingMode  [strong]
```

Markov chain sampling mode.

Enumerator

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.
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.
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 55 of file bm-pbil.hh.

5.9.3 Member Function Documentation

5.9.3.1 set_selection_size()

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

Set the selection size.

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

Definition at line 307 of file bm-pbil.hh.

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

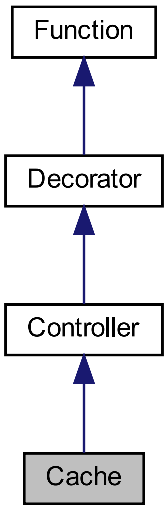
- lib/hnco/algorithms/walsh-moment/bm-pbil.hh

5.10 Cache Class Reference

Cache.

`#include <hnco/functions/controllers/controller.hh>`

Inheritance diagram for Cache:



Public Member Functions

- Cache (Function ∗function)

    Constructor.
- bool provides_incremental_evaluation () const

    Check whether the function provides incremental evaluation.
- double **get_lookup_ratio** ()

    Get lookup ratio.

Evaluation

- double **evaluate** (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

5.10.1 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 369 of file controller.hh.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Cache()

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

Constructor.

Parameters

function	Decorated function

Definition at line 389 of file controller.hh.

5.10.3 Member Function Documentation

5.10.3.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Controller.

Definition at line 399 of file controller.hh.

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

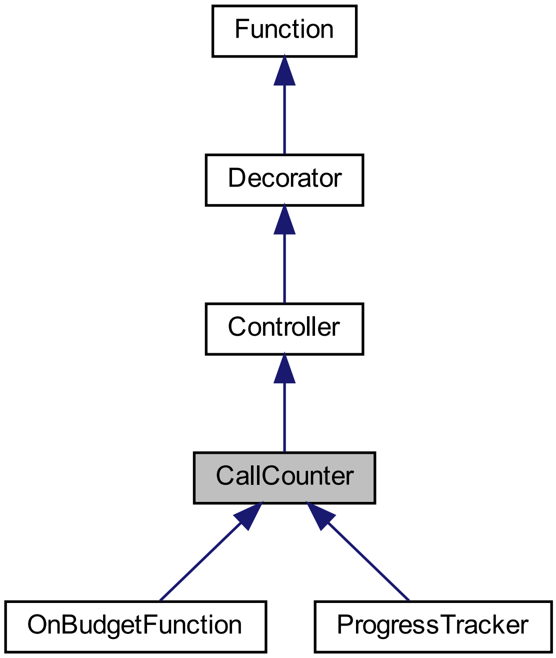
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.11 CallCounter Class Reference

Call counter.

`#include <hnco/functions/controllers/controller.hh>`

Inheritance diagram for CallCounter:



Public Member Functions

- **CallCounter** (Function ∗function)
  Constructor.
- int **get_num_calls** ()
  Get the number of calls.

Evaluation

- double **evaluate** (const bit_vector_t &)
  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &bv, double value, const hnco::sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.
- void **update** (const bit_vector_t &bv, double value)
  Update after a safe evaluation.

Protected Attributes

- int **_num_calls**

  Number of calls.

5.11.1 Detailed Description

Call counter.

Definition at line 157 of file controller.hh.

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

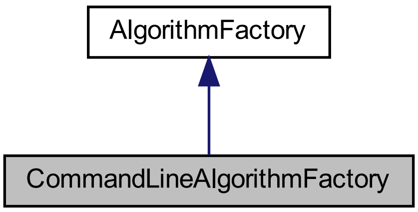
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.12 CommandLineAlgorithmFactory Class Reference

Command line algorithm factory.

`#include <hnco/app/algorithm-factory.hh>`

Inheritance diagram for CommandLineAlgorithmFactory:



Public Member Functions

- **CommandLineAlgorithmFactory** (const HncoOptions &options)

  Constructor.
- hnco::algorithm::Algorithm ∗ make (int bv_size)

  Make an algorithm.

Private Attributes

- const HncoOptions & **_options**

  HNCO options.

5.12.1 Detailed Description

Command line algorithm factory.

Definition at line 42 of file algorithm-factory.hh.

5.12.2 Member Function Documentation

5.12.2.1 make()

```
Algorithm * make (
             int bv_size )  [virtual]
```

Make an algorithm.

Parameters

bv_size	Bit vector size

Implements AlgorithmFactory.

Definition at line 95 of file algorithm-factory.cc.

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

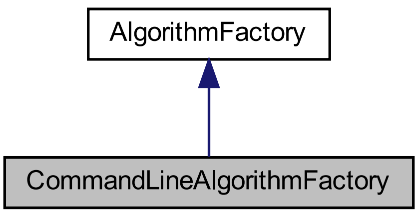
- lib/hnco/app/algorithm-factory.hh
- lib/hnco/app/algorithm-factory.cc

5.13 CommandLineAlgorithmFactory Class Reference

Command line algorithm factory.

```
#include <hnco/multiobjective/app/algorithm-factory.hh>
```

Inheritance diagram for CommandLineAlgorithmFactory:



Public Member Functions

- **CommandLineAlgorithmFactory** (const HncoOptions &options)
  Constructor.
- hnco::multiobjective::algorithm::Algorithm * make (int bv_size, int num_objectives)
  Make an algorithm.

Private Attributes

- const HncoOptions & **_options**
  HNCO options.

5.13.1 Detailed Description

Command line algorithm factory.

Definition at line 46 of file algorithm-factory.hh.

5.13.2 Member Function Documentation

5.13.2.1 make()

```
Algorithm * make (
              int bv_size,
              int num_objectives )  [virtual]
```

Make an algorithm.

Parameters

bv_size	Bit vector size

Implements AlgorithmFactory.

Definition at line 33 of file algorithm-factory.cc.

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

- lib/hnco/multiobjective/app/algorithm-factory.hh
- lib/hnco/multiobjective/app/algorithm-factory.cc

5.14 CommandLineApplication Class Reference

Command line application.

```
#include <hnco/app/application.hh>
```

Public Member Functions

- CommandLineApplication (const HncoOptions &options, FunctionFactory &function_factory, AlgorithmFactory &algorithm_factory)

  Constructor.
- void **run** ()

  Run the application.

Private Member Functions

- void **init** ()

  Initialization.
- void **make_functions** ()

  Make all functions.
- void **load_solution** ()

  Load a solution.
- void **print_information** ()

  Print information about the function.
- void **make_algorithm** ()

  Make algorithm.
- void **maximize** ()

  Maximize the function.
- void **print_results** (double total_time, bool target_reached)

  Print results.
- void **manage_solution** (const bit_vector_t &bv)

  Manage solution.

Private Attributes

- const HncoOptions & **_options**

  HNCO options.
- DecoratedFunctionFactory **_decorated_function_factory**

  Decorated functin factory.
- AlgorithmFactory & **_algorithm_factory**

  Algorithm factory.
- std::vector$<$ function::Function $*$ $>$ **_fns**

  All functions.
- function::Function $*$ **_fn** = nullptr

  Main function.
- hnco::algorithm::Algorithm $*$ **_algorithm** = nullptr

  Algorithm.
- logging::ProgressTrackerContext $*$ **_log_context** = nullptr

  Log context.

5.14.1 Detailed Description

Command line application.

Definition at line 34 of file application.hh.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 CommandLineApplication()

```
CommandLineApplication (
            const HncoOptions & options,
            FunctionFactory & function_factory,
            AlgorithmFactory & algorithm_factory )  [inline]
```

Constructor.

Parameters

options	HNCO options
function_factory	Function factory
algorithm_factory	Algorithm factory

Definition at line 89 of file application.hh.

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

- lib/hnco/app/application.hh
- lib/hnco/app/application.cc

5.15 CommandLineApplication Class Reference

Command line application.

```
#include <hnco/multiobjective/app/application.hh>
```

Public Member Functions

- CommandLineApplication (const HncoOptions &options, FunctionFactory &function_factory, AlgorithmFactory &algorithm_factory)

  Constructor.
- void **run** ()

  Run the application.

Private Member Functions

- void **init** ()
  Initialization.
- void **make_functions** ()
  Make all functions.
- void **print_information** ()
  Print information about the function.
- void **make_algorithm** ()
  Make algorithm.
- void **minimize** ()
  Minimize objective functions.
- void **manage_solutions** ()
  Manage solutions.

Private Attributes

- const HncoOptions & **_options**
  HNCO options.
- FunctionFactory & **_function_factory**
  Functin factory.
- AlgorithmFactory & **_algorithm_factory**
  Algorithm factory.
- std::vector< hnco::multiobjective::function::Function * > **_fns**
  All functions.
- hnco::multiobjective::function::Function * **_fn** = nullptr
  Main function.
- hnco::multiobjective::algorithm::Algorithm * **_algorithm** = nullptr
  Algorithm.
- logging::ProgressTrackerContext * **_log_context** = nullptr
  Log context.

5.15.1 Detailed Description

Command line application.

Definition at line 37 of file application.hh.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 CommandLineApplication()

```
CommandLineApplication (
            const HncoOptions & options,
            FunctionFactory & function_factory,
            AlgorithmFactory & algorithm_factory )  [inline]
```

Constructor.

Parameters

options	HNCO options
function_factory	Function factory
algorithm_factory	Algorithm factory

Definition at line 86 of file application.hh.

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

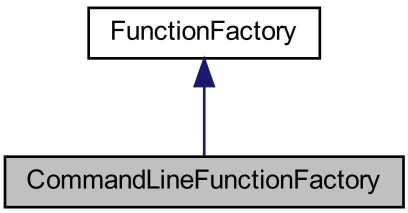
- lib/hnco/multiobjective/app/application.hh
- lib/hnco/multiobjective/app/application.cc

5.16 CommandLineFunctionFactory Class Reference

Command line function factory.

`#include <hnco/app/function-factory.hh>`

Inheritance diagram for CommandLineFunctionFactory:



Public Member Functions

- **CommandLineFunctionFactory** (const HncoOptions &options)
    Constructor.
- hnco::function::Function ∗ **make** ()
    Make a function.

Private Attributes

- const HncoOptions & **_options**
    HNCO options.

5.16.1 Detailed Description

Command line function factory.

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

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

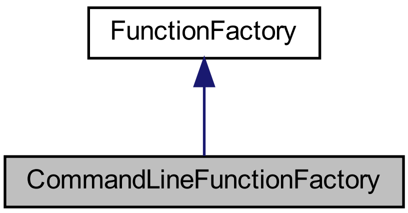
- lib/hnco/app/function-factory.hh
- lib/hnco/app/function-factory.cc

5.17 CommandLineFunctionFactory Class Reference

Command line function factory.

```
#include <hnco/multiobjective/app/function-factory.hh>
```

Inheritance diagram for CommandLineFunctionFactory:



Public Member Functions

- **CommandLineFunctionFactory** (const HncoOptions &options)

    Constructor.
- hnco::multiobjective::function::Function ∗ **make** ()

    Make a function.

Private Attributes

- const HncoOptions & **_options**

    HNCO options.

5.17.1 Detailed Description

Command line function factory.

Definition at line 44 of file function-factory.hh.

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

- lib/hnco/multiobjective/app/function-factory.hh
- lib/hnco/multiobjective/app/function-factory.cc

5.18 CommaSelection Class Reference

Comma selection.

`#include <hnco/algorithms/evolutionary-algorithms/selection.hh>`

Public Member Functions

- CommaSelection (Population &parents, Population &offsprings)
  Constructor.
- void **select** ()
  Apply selection.

Private Attributes

- Population & **_parents**
  Parent population.
- Population & **_offsprings**
  Offspring population.

5.18.1 Detailed Description

Comma selection.

Used as selection for replacement in evolutionary algorithms.

Definition at line 38 of file selection.hh.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 CommaSelection()

```
CommaSelection (
            Population & parents,
            Population & offsprings )  [inline]
```

Constructor.

Parameters

parents	Parent population
offsprings	Offspring population

Definition at line 53 of file selection.hh.

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

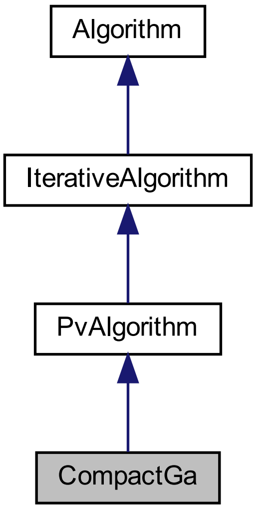
- lib/hnco/algorithms/evolutionary-algorithms/selection.hh

5.19 CompactGa Class Reference

Compact genetic algorithm.

`#include <hnco/algorithms/probability-vector/compact-ga.hh>`

Inheritance diagram for CompactGa:



Public Member Functions

- **CompactGa** (int n)

  Constructor.

Setters

- void **set_learning_rate** (double x)

  Set the learning rate.

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- std::vector< bit_vector_t > **_candidates**
  Candidates.

Parameters

- double **_learning_rate** = 1e-3
  Learning rate.

5.19.1 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 41 of file compact-ga.hh.

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

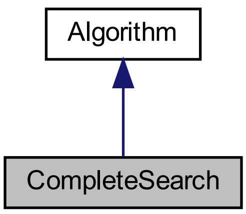
- lib/hnco/algorithms/probability-vector/compact-ga.hh
- lib/hnco/algorithms/probability-vector/compact-ga.cc

5.20 CompleteSearch Class Reference

Complete search.

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

Inheritance diagram for CompleteSearch:



Public Member Functions

- **CompleteSearch** (int n)

  Constructor.
- void **maximize** (const std::vector$<$ function::Function $*$ $>$ &functions)

  Maximize.

Additional Inherited Members

5.20.1 Detailed Description

Complete search.

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

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

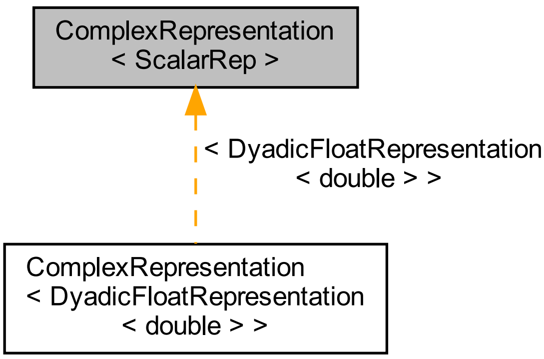
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

5.21 ComplexRepresentation$<$ ScalarRep $>$ Class Template Reference

Complex representation.

`#include <hnco/representations/complex.hh>`

Inheritance diagram for ComplexRepresentation$<$ ScalarRep $>$:



Public Types

- using **scalar_rep** = ScalarRep

    Scalar representation.
- using **scalar_type** = typename scalar_rep::domain_type

    Scalar type.
- using **domain_type** = std::complex< scalar_type >

    Domain type.

Public Member Functions

- ComplexRepresentation (scalar_rep real_part, scalar_rep imaginary_part)

    Constructor.
- ComplexRepresentation (scalar_rep rep)

    Constructor.
- int **size** () const

    Size of the representation.
- domain_type **unpack** (const bit_vector_t &bv, int start)

    Unpack bit vector into a value.
- void **display** (std::ostream &stream) const

    Display.

Private Attributes

- scalar_rep **_real_part**

    Representation of the real part.
- scalar_rep **_imaginary_part**

    Representation of the imaginary part.

5.21.1 Detailed Description

template<class ScalarRep>
class hnco::representation::ComplexRepresentation< ScalarRep >

Complex representation.

Definition at line 39 of file complex.hh.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ComplexRepresentation() [1/2]

```
ComplexRepresentation (
            scalar_rep real_part,
            scalar_rep imaginary_part )   [inline]
```

Constructor.

Parameters

real_part	Representation of real part
imaginary_part	Representation of imaginary part

Definition at line 68 of file complex.hh.

5.21.2.2 ComplexRepresentation() [2/2]

```
ComplexRepresentation (
            scalar_rep rep )  [inline]
```

Constructor.

Parameters

rep	Representation of both real and imaginary parts

Definition at line 78 of file complex.hh.

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

- lib/hnco/representations/complex.hh

5.22 ComplexToDouble< T > Struct Template Reference

Convert a complex to a double.

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

Public Types

- using **codomain_type** = std::complex< T >
  Codomain type.

Public Member Functions

- double **operator()** (std::complex< T > z)
  Convert to double.

5.22.1 Detailed Description

template<class T>
struct hnco::function::ComplexToDouble< T >

Convert a complex to a double.

Definition at line 44 of file converter.hh.

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

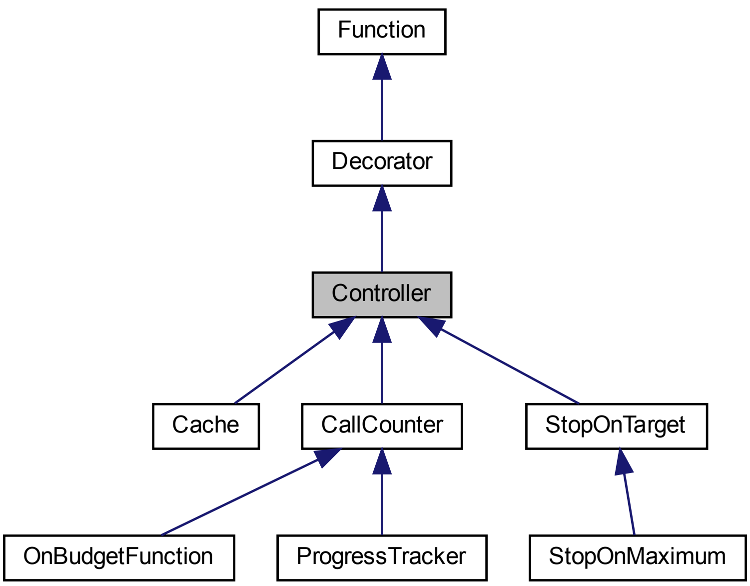
- lib/hnco/functions/converter.hh

5.23 Controller Class Reference

Function controller.

`#include <hnco/functions/controllers/controller.hh>`

Inheritance diagram for Controller:



Public Member Functions

- **Controller** (Function ∗function)
  Constructor.

Information about the function

- int **get_bv_size** () const
  Get bit vector size.
- double **get_maximum** () const
  Get the global maximum.
- bool **has_known_maximum** () const
  Check for a known maximum.
- bool provides_incremental_evaluation () const
  Check whether the function provides incremental evaluation.

Evaluation

- double **evaluate_safely** (const bit_vector_t &bv)
  Safely evaluate a bit vector.

Additional Inherited Members

5.23.1 Detailed Description

Function controller.

Definition at line 41 of file controller.hh.

5.23.2 Member Function Documentation

5.23.2.1 provides_incremental_evaluation()

```
bool provides_incremental_evaluation ( ) const  [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 67 of file controller.hh.

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

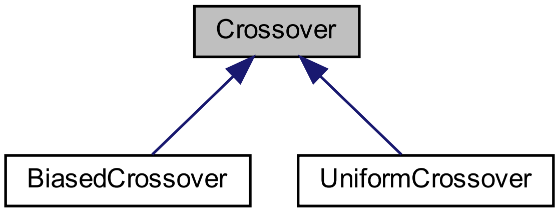
- lib/hnco/functions/controllers/controller.hh

5.24 Crossover Class Reference

Crossover

```
#include <hnco/algorithms/evolutionary-algorithms/crossover.hh>
```

Inheritance diagram for Crossover:



Public Member Functions

- virtual ~**Crossover** ()
  Destructor.
- virtual void recombine (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)=0
  Recombine.

5.24.1 Detailed Description

Crossover

Definition at line 35 of file crossover.hh.

5.24.2 Member Function Documentation

5.24.2.1 recombine()

```
virtual void recombine (
            const bit_vector_t & parent1,
            const bit_vector_t & parent2,
            bit_vector_t & offspring )  [pure virtual]
```

Recombine.

The offspring is the crossover of two parents.

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implemented in UniformCrossover, and BiasedCrossover.

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

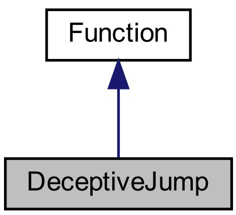
- lib/hnco/algorithms/evolutionary-algorithms/crossover.hh

5.25 DeceptiveJump Class Reference

Deceptive jump.

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

Inheritance diagram for DeceptiveJump:



Public Member Functions

- **DeceptiveJump** (int bv_size, int gap)

  Constructor.
- int **get_bv_size** () const override

  Get bit vector size.
- bool has_known_maximum () const override

  Check for a known maximum.
- double get_maximum () const override

  Get the global maximum.
- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.

Private Attributes

- int **_bv_size**

  Bit vector size.
- int **_gap**

  Gap.

5.25.1 Detailed Description

Deceptive jump.

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

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 85 of file jump.hh.

5.25.2 Member Function Documentation

5.25.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size + _gap

Reimplemented from Function.

Definition at line 108 of file jump.hh.

5.25.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 104 of file jump.hh.

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

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

5.26 DecoratedFunctionFactory Class Reference

Decorated function factory.

`#include <hnco/app/decorated-function-factory.hh>`

Public Member Functions

- **DecoratedFunctionFactory** (const HncoOptions &options, FunctionFactory &function_factory)
  Constructor.
- hnco::function::Function ∗ **make_function_modifier** ()
  Make a function modifier.
- hnco::function::Function ∗ make_function_controller (hnco::function::Function ∗function)
  Make a function controller.
- hnco::map::Map ∗ **get_map** ()
  Get map.
- hnco::function::controller::ProgressTracker ∗ **get_tracker** ()
  Get tracker controller.
- hnco::function::controller::Cache ∗ **get_cache** ()
  Get Cache controller.
- hnco::function::controller::StopOnTarget ∗ **get_stop_on_target** ()
  Get StopOnTarget controller.

Private Member Functions

- hnco::function::Function ∗ **make_function** ()
  Make a function.

Private Attributes

- const HncoOptions & **_options**
  HNCO options.
- FunctionFactory & **_function_factory**
  Factory function.
- hnco::map::Map ∗ **_map** = nullptr
  Map.
- hnco::function::controller::ProgressTracker ∗ **_tracker** = nullptr
  Tracker controller.
- hnco::function::controller::Cache ∗ **_cache** = nullptr
  Cache controller.
- hnco::function::controller::StopOnTarget ∗ **_stop_on_target** = nullptr
  StopOnTarget controller.

5.26.1 Detailed Description

Decorated function factory.

Definition at line 35 of file decorated-function-factory.hh.

5.26.2 Member Function Documentation

5.26.2.1 make_function_controller()

```
Function * make_function_controller (
            hnco::function::Function * function )
```

Make a function controller.

Parameters

function	Decorated function

Definition at line 257 of file decorated-function-factory.cc.

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

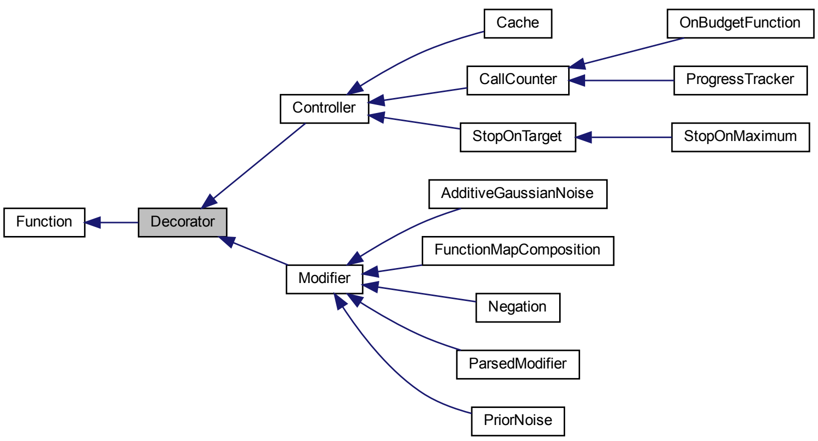
- lib/hnco/app/decorated-function-factory.hh
- lib/hnco/app/decorated-function-factory.cc

5.27 Decorator Class Reference

Function decorator

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

Inheritance diagram for Decorator:



Public Member Functions

- **Decorator** (Function *function)
  Constructor.

Display

- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &x, std::ostream &stream) override
  Describe a bit vector.

Protected Attributes

- Function * **_function**
  Decorated function.

5.27.1 Detailed Description

Function decorator

Definition at line 34 of file decorator.hh.

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

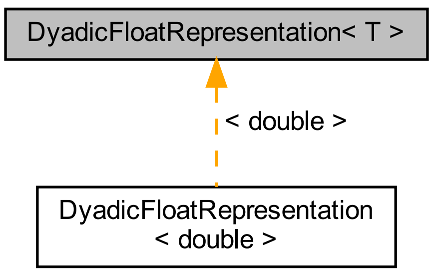
- lib/hnco/functions/decorator.hh

5.28 DyadicFloatRepresentation$<$ T $>$ Class Template Reference

Dyadic float representation.

```
#include <hnco/representations/float.hh>
```

Inheritance diagram for DyadicFloatRepresentation$<$ T $>$:



Public Types

- using **domain_type** = T

  Domain type.

Public Member Functions

- DyadicFloatRepresentation (T lower_bound, T upper_bound, int size)

  Constructor.
- DyadicFloatRepresentation (T lower_bound, T upper_bound, T precision)

  Constructor.
- int **size** () const

  Size of the representation.
- domain_type **unpack** (const bit_vector_t &bv, int start)

  Unpack bit vector into a value.
- void **display** (std::ostream &stream) const

  Display.

Private Member Functions

- T **affine_transformation** (T x)

  Affine transformation.
- void compute_lengths (int size)

  Compute lengths.

Private Attributes

- std::vector< T > **_lengths**

  Lengths of dyadic intervals.
- T **_lower_bound**

  Lower bound of the interval.
- T **_length**

  Length of the interval.

5.28.1 Detailed Description

template<class T>
class hnco::representation::DyadicFloatRepresentation< T >

Dyadic float representation.

Definition at line 44 of file float.hh.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 DyadicFloatRepresentation() [1/2]

```
DyadicFloatRepresentation (
            T lower_bound,
            T upper_bound,
            int size )  [inline]
```

Constructor.

The represented interval is [lower_bound, upper_bound).

Parameters

lower_bound	Lower bound of the interval
upper_bound	Upper bound of the interval
size	Size in bits per float number

Definition at line 89 of file float.hh.

5.28.2.2 DyadicFloatRepresentation() [2/2]

```
DyadicFloatRepresentation (
            T lower_bound,
            T upper_bound,
            T precision )  [inline]
```

Constructor.

The represented interval is [lower_bound, upper_bound).

Parameters

lower_bound	Lower bound of the interval
upper_bound	Upper bound of the interval
precision	Precision

Definition at line 108 of file float.hh.

5.28.3 Member Function Documentation

5.28.3.1 compute_lengths()

```
void compute_lengths (
            int size )  [inline], [private]
```

Compute lengths.

Parameters

size	Size in bits per float number

Definition at line 63 of file float.hh.

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

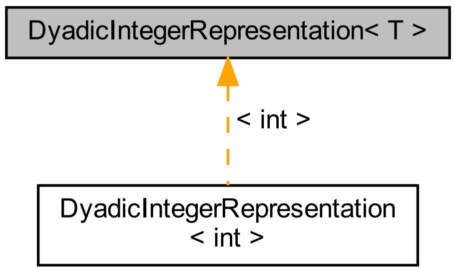
- lib/hnco/representations/float.hh

5.29 DyadicIntegerRepresentation< T > Class Template Reference

Dyadic integer representation.

```
#include <hnco/representations/integer.hh>
```

Inheritance diagram for DyadicIntegerRepresentation< T >:



Classes

- struct Precision

  Precision

Public Types

- using **domain_type** = T

  Domain type.

Public Member Functions

- DyadicIntegerRepresentation (T lower_bound, T upper_bound, int size)

  Constructor.
- DyadicIntegerRepresentation (T lower_bound, T upper_bound)

  Constructor.
- DyadicIntegerRepresentation (T lower_bound, T upper_bound, Precision precision)

  Constructor.
- int **size** () const

  Size of the representation.
- domain_type **unpack** (const bit_vector_t &bv, int start)

  Unpack bit vector into a value.
- void **display** (std::ostream &stream) const

  Display.

Private Member Functions

- void **set_exact_size** (T lower_bound, T upper_bound)

  Set the exact size for a given interval.

Private Attributes

- int **_size**

  Size in bits.
- int **_exact_size**

  Exact size required for a given interval.
- T **_lower_bound**

  Lower bound of the interval.
- T **_upper_bound**

  Upper bound of the interval.

5.29.1 Detailed Description

template<class T>
class hnco::representation::DyadicIntegerRepresentation< T >

Dyadic integer representation.

Definition at line 73 of file integer.hh.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 DyadicIntegerRepresentation() [1/3]

```
DyadicIntegerRepresentation (
            T lower_bound,
            T upper_bound,
            int size ) [inline]
```

Constructor.

The represented interval is [lower_bound..upper_bound].

Parameters

lower_bound	Lower bound of the interval
upper_bound	Upper bound of the interval
size	Size in bits per integer

Definition at line 121 of file integer.hh.

5.29.2.2 DyadicIntegerRepresentation() [2/3]

```
DyadicIntegerRepresentation (
            T lower_bound,
            T upper_bound )  [inline]
```

Constructor.

The represented interval is [lower_bound..upper_bound].

Parameters

lower_bound	Lower bound of the interval
upper_bound	Upper bound of the interval

Definition at line 142 of file integer.hh.

5.29.2.3 DyadicIntegerRepresentation() [3/3]

```
DyadicIntegerRepresentation (
            T lower_bound,
            T upper_bound,
            Precision precision )  [inline]
```

Constructor.

The represented interval is [lower_bound..upper_bound].

Parameters

lower_bound	Lower bound of the interval
upper_bound	Upper bound of the interval
precision	Precision

Definition at line 159 of file integer.hh.

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

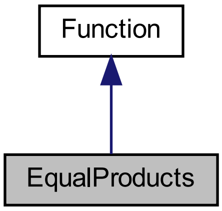
- lib/hnco/representations/integer.hh

5.30 EqualProducts Class Reference

Equal products.

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

Inheritance diagram for EqualProducts:



Public Member Functions

- **EqualProducts** ()
    Constructor.
- int **get_bv_size** () const override
    Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
    Evaluate a bit vector.

Instance generators

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

Load and save instance

- void load (std::string path)
    Load instance.
- void save (std::string path) const
    Save instance.

Private Member Functions

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

Private Attributes

- std::vector< double > **_numbers**
    Numbers.

5.30.1 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 59 of file equal-products.hh.

5.30.2 Member Function Documentation

5.30.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Number generator

Definition at line 91 of file equal-products.hh.

5.30.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 124 of file equal-products.hh.

5.30.2.3 random()

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

Random instance.

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

Parameters

n	Size of bit vector

Definition at line 106 of file equal-products.hh.

5.30.2.4 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 131 of file equal-products.hh.

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

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

5.31 ProgressTracker::Event Struct Reference

Event

`#include <hnco/functions/controllers/controller.hh>`

Public Attributes

- int **num_evaluations**
  Number of evaluations.
- algorithm::solution_t **solution**
  Solution.

5.31.1 Detailed Description

Event

Definition at line 246 of file controller.hh.

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

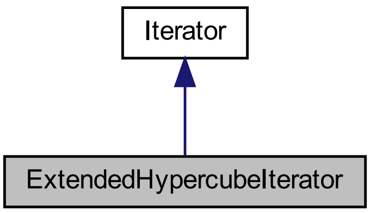
- lib/hnco/functions/controllers/controller.hh

5.32 ExtendedHypercubeIterator Class Reference

Extended Hypercube iterator.

`#include <hnco/iterator.hh>`

Inheritance diagram for ExtendedHypercubeIterator:



Public Member Functions

- **ExtendedHypercubeIterator** (int n)

    Constructor.
- bool **has_next** () override

    Has next bit vector.
- const bit_vector_t & **next** () override

    Next bit vector.

Additional Inherited Members

5.32.1 Detailed Description

Extended Hypercube iterator.

Similar to Hypercube. In dimension 0, an HypercubeIterator does not contain any element. However, in dimension 0, an ExtendedHypercubeIterator contains a unique element which is the vector of size 0. An ExtendedHypercubeIterator is helpful when the enumerated vectors are seen as prefixes or suffixes hence can be empty. This is used, in particular, in compute_fast_walsh_transform.

Definition at line 97 of file iterator.hh.

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

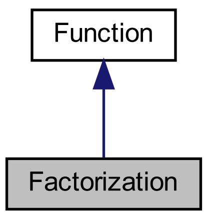
- lib/hnco/iterator.hh
- lib/hnco/iterator.cc

5.33 Factorization Class Reference

Factorization.

```
#include <hnco/functions/collection/factorization.hh>
```

Inheritance diagram for Factorization:



Public Member Functions

- **Factorization** ()
  Constructor.
- Factorization (const std::string number)
  Constructor.
- ∼**Factorization** ()
  Destructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &x, std::ostream &stream) override
  Describe a bit vector.

Load and save instance

- void load (std::string path)
  Load instance.

Private Member Functions

- void **init** ()
  Init GMP data structures.
- void **clear** ()
  Clear GMP data structures.
- void **set_number** (const std::string number)
  Set number.
- void **convert** (const bit_vector_t &x)
  Convert a bit vector into two numbers.

Private Attributes

- mpz_t **_number**
  Number to factorize.
- mpz_t **_first_factor**
  First factor.
- mpz_t **_second_factor**
  Second factor.
- mpz_t **_product**
  Product.
- std::string **_first_factor_string**
  First factor in binary form.
- std::string **_second_factor_string**
  Secon factor in binary form.
- size_t **_number_size**
  Number size in bits.
- size_t **_first_factor_size**
  First factor size in bits.
- size_t **_second_factor_size**
  Second factor size in bits.
- int **_bv_size**
  Bit vector size.

5.33.1 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 29 of file factorization.hh.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 Factorization()

```
Factorization (
            const std::string number )  [inline]
```

Constructor.

Parameters

number	Number to factorize written in decimal form

Definition at line 82 of file factorization.hh.

5.33.3 Member Function Documentation

5.33.3.1 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

The file referenced by the path is a text file which contains exactly one natural number written in base 10 without any space

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 102 of file factorization.hh.

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

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

5.34 FfgenOptions Class Reference

Command line options for ffgen.

```
#include <ffgen-options.hh>
```

Public Member Functions

- **FfgenOptions** ()
  Default constructor.
- **FfgenOptions** (int argc, char ∗argv[ ], bool ignore_bad_options=false)
  Constructor.
- int **get_bv_size** () const
  Get the value of bv_size.
- bool **with_bv_size** () const
  With parameter bv_size.
- double **get_coupling_constant** () const
  Get the value of coupling_constant.
- bool **with_coupling_constant** () const
  With parameter coupling_constant.
- double **get_ep_upper_bound** () const
  Get the value of ep_upper_bound.
- bool **with_ep_upper_bound** () const
  With parameter ep_upper_bound.
- double **get_field_constant** () const
  Get the value of field_constant.
- bool **with_field_constant** () const
  With parameter field_constant.
- int **get_function** () const
  Get the value of function.
- bool **with_function** () const
  With parameter function.
- double **get_lin_distance** () const
  Get the value of lin_distance.
- bool **with_lin_distance** () const
  With parameter lin_distance.
- int **get_lin_generator** () const

Get the value of lin_generator.
- bool **with_lin_generator** () const
  With parameter lin_generator.
- double **get_lin_initial_weight** () const
  Get the value of lin_initial_weight.
- bool **with_lin_initial_weight** () const
  With parameter lin_initial_weight.
- double **get_lin_ratio** () const
  Get the value of lin_ratio.
- bool **with_lin_ratio** () const
  With parameter lin_ratio.
- int **get_ms_num_clauses** () const
  Get the value of ms_num_clauses.
- bool **with_ms_num_clauses** () const
  With parameter ms_num_clauses.
- int **get_ms_num_literals_per_clause** () const
  Get the value of ms_num_literals_per_clause.
- bool **with_ms_num_literals_per_clause** () const
  With parameter ms_num_literals_per_clause.
- int **get_nk_k** () const
  Get the value of nk_k.
- bool **with_nk_k** () const
  With parameter nk_k.
- int **get_nn1_generator** () const
  Get the value of nn1_generator.
- bool **with_nn1_generator** () const
  With parameter nn1_generator.
- int **get_nn2_generator** () const
  Get the value of nn2_generator.
- bool **with_nn2_generator** () const
  With parameter nn2_generator.
- int **get_nn2_num_columns** () const
  Get the value of nn2_num_columns.
- bool **with_nn2_num_columns** () const
  With parameter nn2_num_columns.
- int **get_nn2_num_rows** () const
  Get the value of nn2_num_rows.
- bool **with_nn2_num_rows** () const
  With parameter nn2_num_rows.
- int **get_part_upper_bound** () const
  Get the value of part_upper_bound.
- bool **with_part_upper_bound** () const
  With parameter part_upper_bound.
- std::string **get_path** () const
  Get the value of path.
- bool **with_path** () const
  With parameter path.
- int **get_seed** () const
  Get the value of seed.
- bool **with_seed** () const
  With parameter seed.

- double **get_stddev** () const

  Get the value of stddev.
- bool **with_stddev** () const

  With parameter stddev.
- int **get_sudoku_num_empty_cells** () const

  Get the value of sudoku_num_empty_cells.
- bool **with_sudoku_num_empty_cells** () const

  With parameter sudoku_num_empty_cells.
- int **get_walsh2_generator** () const

  Get the value of walsh2_generator.
- bool **with_walsh2_generator** () const

  With parameter walsh2_generator.
- double **get_walsh2_ising_alpha** () const

  Get the value of walsh2_ising_alpha.
- bool **with_walsh2_ising_alpha** () const

  With parameter walsh2_ising_alpha.
- int **get_walsh_num_features** () const

  Get the value of walsh_num_features.
- bool **with_walsh_num_features** () const

  With parameter walsh_num_features.
- bool **with_ms_planted_solution** () const

  With the flag ms_planted_solution.
- bool **with_periodic_boundary_conditions** () const

  With the flag periodic_boundary_conditions.

Private Member Functions

- void **print_help** (std::ostream &stream) const

  Print help message.
- void **print_version** (std::ostream &stream) const

  Print version.

Private Attributes

- std::string **_exec_name**

  Name of the executable.
- std::string **_version** = "0.24"

  Name Version.
- int **_bv_size** = 100

  Size of bit vectors.
- double **_coupling_constant** = 1

  Coupling constant.
- double **_ep_upper_bound** = 1

  Upper bound of numbers.
- double **_field_constant** = 1

  Field constant.
- int **_function** = 1

  Type of function.
- double **_lin_distance** = 1

Common distance of arithmetic progression.

- int **_lin_generator** = 0

  Type of LinearFunction generator.
- double **_lin_initial_weight** = 1

  Initial weight.
- double **_lin_ratio** = 2

  Common ratio of geometric progression.
- int **_ms_num_clauses** = 100

  Number of clauses.
- int **_ms_num_literals_per_clause** = 3

  Number of literals per clause.
- int **_nk_k** = 3

  Each bit is connected to k other bits.
- int **_nn1_generator** = 0

  Type of NearestNeighborIsingModel1 generator.
- int **_nn2_generator** = 0

  Type of NearestNeighborIsingModel2 generator.
- int **_nn2_num_columns** = 10

  Number of columns.
- int **_nn2_num_rows** = 10

  Number of rows.
- int **_part_upper_bound** = 100

  Upper bound of numbers.
- std::string **_path** = "function.txt"

  Path (relative or absolute) of a function file.
- int **_seed**

  Seed for the random number generator.
- double **_stddev** = 1

  Standard deviation.
- int **_sudoku_num_empty_cells** = 10

  Number of empty cells.
- int **_walsh2_generator** = 0

  Type of WalshExpansion2 generator.
- double **_walsh2_ising_alpha** = 2

  Dyson-Ising: exponential decay parameter for long range interactions.
- int **_walsh_num_features** = 100

  Number of features.
- bool **_ms_planted_solution** = false

  Generate an instance with a planted solution.
- bool **_periodic_boundary_conditions** = false

  Periodic boundary conditions.

Friends

- std::ostream & **operator**<< (std::ostream &, const FfgenOptions &)

  Print a header containing the parameter values.

5.34.1 Detailed Description

Command line options for ffgen.

Definition at line 11 of file ffgen-options.hh.

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

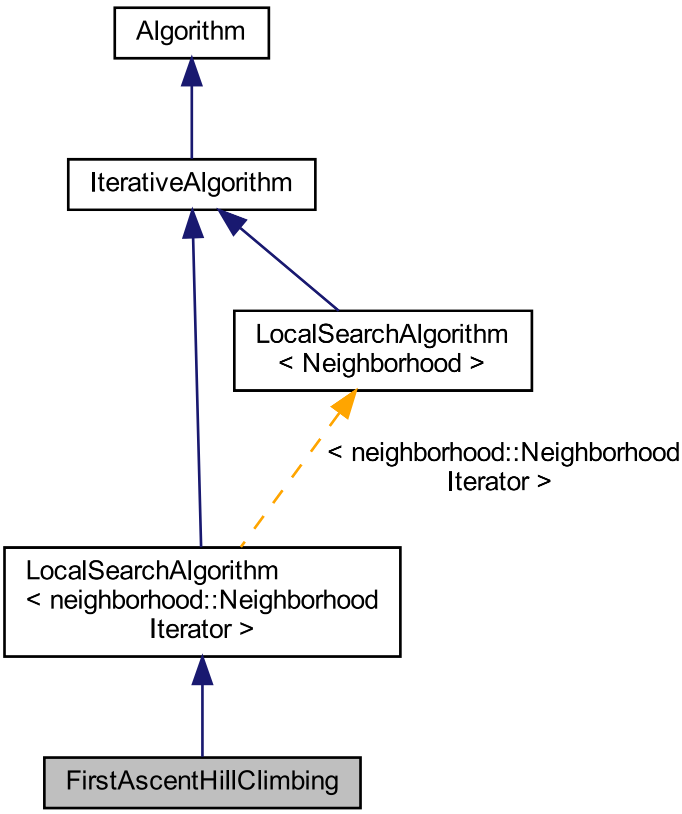
- app/ffgen-options.hh
- app/ffgen-options.cc

5.35 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

```
#include <hnco/algorithms/local-search/first-ascent-hill-climbing.hh>
```

Inheritance diagram for FirstAscentHillClimbing:



Public Member Functions

- **FirstAscentHillClimbing** (int n, neighborhood::NeighborhoodIterator ∗neighborhood)

  Constructor.

Protected Member Functions

- void **iterate** () override

  Single iteration.

Additional Inherited Members

5.35.1 Detailed Description

First ascent hill climbing.

Definition at line 34 of file first-ascent-hill-climbing.hh.

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

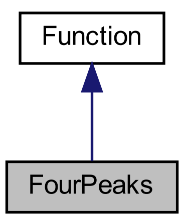
- lib/hnco/algorithms/local-search/first-ascent-hill-climbing.hh
- lib/hnco/algorithms/local-search/first-ascent-hill-climbing.cc

5.36 FourPeaks Class Reference

Four Peaks.

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

Inheritance diagram for FourPeaks:



Public Member Functions

- **FourPeaks** (int bv_size, int threshold)

    Constructor.
- int **get_bv_size** () const override

    Get bit vector size.
- bool has_known_maximum () const override

    Check for a known maximum.
- double get_maximum () const override

    Get the global maximum.
- double **evaluate** (const bit_vector_t &) override

    Evaluate a bit vector.

Private Attributes

- int **_bv_size**

    Bit vector size.
- int **_threshold**

    Threshold.
- int **_maximum**

    Maximum.

5.36.1 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.

5.36.2 Member Function Documentation

5.36.2.1 get_maximum()

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

Get the global maximum.

Returns

2 ∗ _bv_size - _threshold - 1

Reimplemented from Function.

Definition at line 88 of file four-peaks.hh.

5.36.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 84 of file four-peaks.hh.

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

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

5.37 FrontDistancePair Struct Reference

Front-distance pair.

```
#include <hnco/multiobjective/algorithms/nsga2.hh>
```

Public Attributes

- int **pareto_front**
  Pareto front.
- double **crowding_distance**
  Crowding distance.

5.37.1 Detailed Description

Front-distance pair.

A front-distance pair measures the quality of an individual within a population.

Definition at line 45 of file nsga2.hh.

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

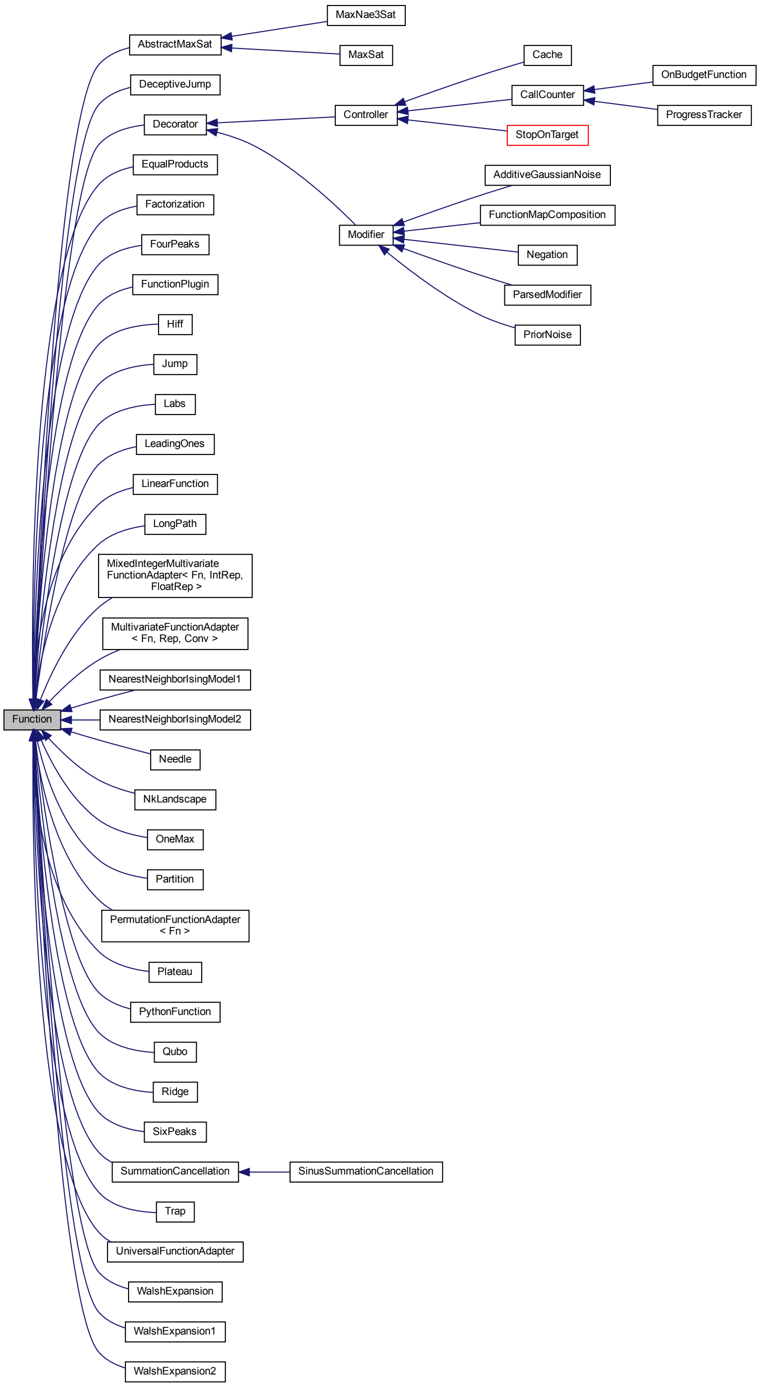
- lib/hnco/multiobjective/algorithms/nsga2.hh

5.38 Function Class Reference

Function

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

Inheritance diagram for Function:



Public Member Functions

- virtual ∼**Function** ()

  Destructor.

Information about the function

- virtual int **get_bv_size** () const =0
  Get bit vector size.
- virtual double get_maximum () const
  Get the global maximum.
- virtual bool **has_known_maximum** () const
  Check for a known maximum.
- virtual bool provides_incremental_evaluation () const
  Check whether the function provides incremental evaluation.

Evaluation

- virtual double evaluate (const bit_vector_t &)=0
  Evaluate a bit vector.
- virtual double evaluate_incrementally (const bit_vector_t &x, double value, const sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.
- virtual double evaluate_safely (const bit_vector_t &x)
  Safely evaluate a bit vector.
- virtual void update (const bit_vector_t &x, double value)
  Update states after a safe evaluation.

Display

- virtual void **display** (std::ostream &stream) const
  Display.
- virtual void describe (const bit_vector_t &x, std::ostream &stream)
  Describe a bit vector.

5.38.1 Detailed Description

Function

Definition at line 41 of file function.hh.

5.38.2 Member Function Documentation

5.38.2.1 describe()

```
virtual void describe (
            const bit_vector_t & x,
            std::ostream & stream )  [inline], [virtual]
```

Describe a bit vector.

The member function Function::describe is not declared const for the same reason Function::evaluate is not: it might need to decode the given bit vector hence use some pre-allocated memory buffer.

Reimplemented in MultivariateFunctionAdapter< Fn, Rep, Conv >, MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep PermutationFunctionAdapter< Fn >, UniversalFunctionAdapter, Factorization, Partition, Decorator, and FunctionMapComposition.

Definition at line 130 of file function.hh.

5.38.2.2 evaluate()

```
virtual double evaluate (
            const bit_vector_t & )  [pure virtual]
```

Evaluate a bit vector.

This member function is not declared const and is not supposed to be thread-safe. In particular, in order to evaluate a bit vector, it might require some data member to store temporary results. In case of parallel evaluation, there should be a copy of the function per thread, as is done in Population::evaluate_in_parallel.

Implemented in LongPath, FunctionPlugin, Trap, StopOnTarget, CallCounter, OnBudgetFunction, ProgressTracker, Cache, EqualProducts, Factorization, FourPeaks, SixPeaks, NearestNeighborIsingModel1, NearestNeighborIsingModel2, Jump, DeceptiveJump, Labs, LinearFunction, MaxSat, MaxNae3Sat, NkLandscape, Partition, PythonFunction, Qubo, OneMax, LeadingOnes, Needle, Hiff, Ridge, Plateau, WalshExpansion1, WalshExpansion2, WalshExpansion, Negation, FunctionMapComposition, AdditiveGaussianNoise, ParsedModifier, PriorNoise, MultivariateFunctionAdapter< Fn, Rep, Co MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep >, PermutationFunctionAdapter< Fn >, UniversalFunctionAdapter, SummationCancellation, and SinusSummationCancellation.

5.38.2.3 evaluate_incrementally()

```
virtual double evaluate_incrementally (
            const bit_vector_t & x,
            double value,
            const sparse_bit_vector_t & flipped_bits )  [inline], [virtual]
```

Incrementally evaluate a bit vector.

Exceptions

std::runtime_error	

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

Definition at line 91 of file function.hh.

5.38.2.4 evaluate_safely()

```
virtual double evaluate_safely (
            const bit_vector_t & x )  [inline], [virtual]
```

Safely evaluate a bit vector.

Must neither throw any exception nor update global states (e.g. maximum) in function controllers. It is used in Population::evaluate_in_parallel inside a OMP parallel for loop.

By default, calls evaluate.

Reimplemented in Controller.

Definition at line 105 of file function.hh.

5.38.2.5 get_maximum()

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

Get the global maximum.

Exceptions

std::runtime_error	

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

Definition at line 57 of file function.hh.

5.38.2.6 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented in Controller, Cache, NearestNeighborIsingModel1, NearestNeighborIsingModel2, LinearFunction, OneMax, WalshExpansion1, Negation, and PriorNoise.

Definition at line 67 of file function.hh.

5.38.2.7 update()

```
virtual void update (
            const bit_vector_t & x,
            double value ) [inline], [virtual]
```

Update states after a safe evaluation.

By default, does nothing.

Reimplemented in StopOnTarget, CallCounter, OnBudgetFunction, and ProgressTracker.

Definition at line 111 of file function.hh.

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

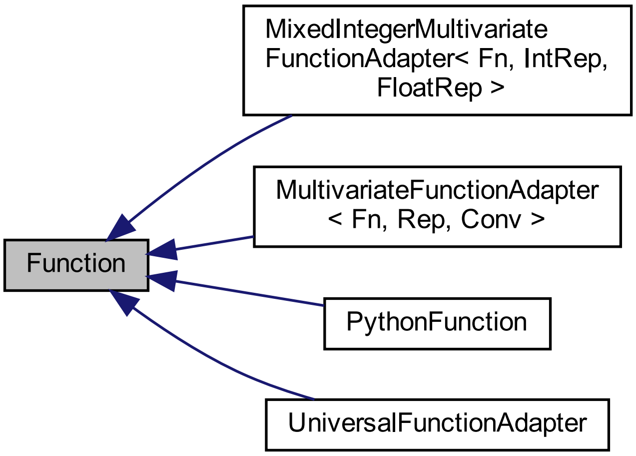
- lib/hnco/functions/function.hh

5.39 Function Class Reference

Function

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

Inheritance diagram for Function:



Public Member Functions

- virtual ∼**Function** ()

  Destructor.

Information about the function

- virtual int **get_bv_size** () const =0

  Get bit vector size.
- virtual int **get_output_size** () const =0

  Get output size (number of objectives)

Evaluation

- virtual void evaluate (const bit_vector_t &bv, value_t &value)=0

  Evaluate a bit vector.

Display

- virtual void **display** (std::ostream &stream) const

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

  Describe a bit vector.

5.39.1 Detailed Description

Function

Definition at line 41 of file function.hh.

5.39.2 Member Function Documentation

5.39.2.1 describe()

```
virtual void describe (
            const bit_vector_t & x,
            std::ostream & stream ) [inline], [virtual]
```

Describe a bit vector.

The member function describe() is not declared const for the same reason evaluate() is not: it might need to decode the given bit vector hence use some pre-allocated memory buffer.

Reimplemented in MultivariateFunctionAdapter< Fn, Rep, Conv >, MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep
and UniversalFunctionAdapter.

Definition at line 95 of file function.hh.

5.39.2.2 evaluate()

```
virtual void evaluate (
            const bit_vector_t & bv,
            value_t & value ) [pure virtual]
```

Evaluate a bit vector.

This member function is not declared const and is not supposed to be thread-safe. In particular, in order to evaluate a bit vector, it might require some data member to store temporary results. In case of parallel evaluation, there should be a copy of the function per thread, as is done in Population::evaluate_in_parallel().

Parameters

bv	Bit vector to evaluate
value	Output value

Implemented in PythonFunction, MultivariateFunctionAdapter< Fn, Rep, Conv >, MixedIntegerMultivariateFunctionAdapter< Fn, IntF
and UniversalFunctionAdapter.

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

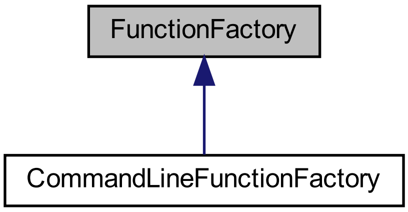
- lib/hnco/multiobjective/functions/function.hh

5.40 FunctionFactory Class Reference

Function factory.

`#include <hnco/app/function-factory.hh>`

Inheritance diagram for FunctionFactory:



Public Member Functions

- virtual hnco::function::Function ∗ **make** ()=0
  Make a function.

5.40.1 Detailed Description

Function factory.

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

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

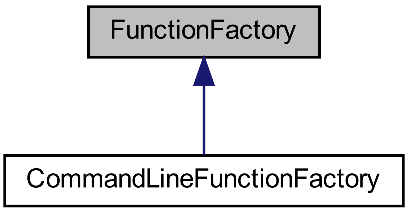
- lib/hnco/app/function-factory.hh

5.41 FunctionFactory Class Reference

Function factory.

`#include <hnco/multiobjective/app/function-factory.hh>`

Inheritance diagram for FunctionFactory:



Public Member Functions

- virtual hnco::multiobjective::function::Function ∗ **make** ()=0

    Make a function.

5.41.1 Detailed Description

Function factory.

Definition at line 36 of file function-factory.hh.

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

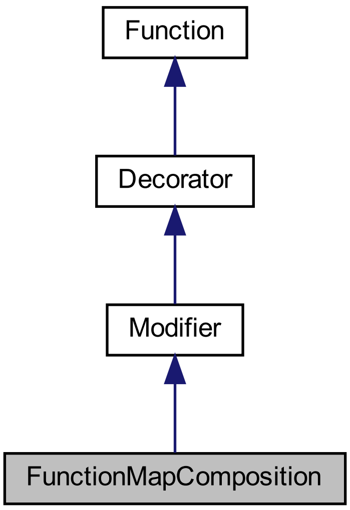
- lib/hnco/multiobjective/app/function-factory.hh

5.42 FunctionMapComposition Class Reference

Composition of a function and a map.

```
#include <hnco/functions/modifiers/modifier.hh>
```

Inheritance diagram for FunctionMapComposition:



Public Member Functions

- FunctionMapComposition (Function ∗function, hnco::map::Map ∗map)

  Constructor.
- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override

  Get bit vector size.
- double get_maximum () const override

  Get the global maximum.
- bool has_known_maximum () const override

  Check for a known maximum.

Display

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

  Describe a bit vector.

Private Attributes

- hnco::map::Map ∗ **_map**

  Map.
- bit_vector_t **_bv**

  Image of bit vectors under the map.

Additional Inherited Members

5.42.1 Detailed Description

Composition of a function and a map.

Definition at line 100 of file modifier.hh.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 FunctionMapComposition()

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

Constructor.

Precondition

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

Exceptions

std::runtime_error	

Definition at line 115 of file modifier.hh.

5.42.3 Member Function Documentation

5.42.3.1 get_maximum()

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

Get the global maximum.

Exceptions

std::runtime_error	

Reimplemented from Function.

Definition at line 135 of file modifier.hh.

5.42.3.2 has_known_maximum()

```
bool has_known_maximum ( ) const  [inline], [override], [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 145 of file modifier.hh.

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

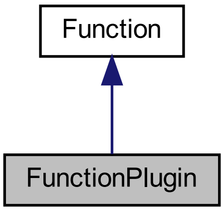
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.43 FunctionPlugin Class Reference

Function plugin

```
#include <hnco/functions/collection/plugin.hh>
```

Inheritance diagram for FunctionPlugin:



Public Member Functions

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

  Constructor.
- ∼**FunctionPlugin** ()

  Destructor.
- int **get_bv_size** () const

  Get bit vector size.
- double **evaluate** (const bit_vector_t &)

  Evaluate a bit vector.

Private Types

- using **extern_function_t** = double(∗)(const bit_t ∗, size_t)

  Type of an extern function.

Private Attributes

- int **_bv_size**

  Bit vector size.
- void ∗ **_handle**

  Handle returned by dlopen.
- extern_function_t **_extern_function**

  Extern function.

5.43.1 Detailed Description

Function plugin

Definition at line 34 of file plugin.hh.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 FunctionPlugin()

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

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

5.44 Generator Struct Reference

Random number generator.

`#include <hnco/random.hh>`

Static Public Member Functions

- static void **set_seed** (unsigned n)
  Set seed.
- static void set_seed ()
  Set seed.
- static void reset ()
  Reset engine.
- static double **uniform** ()
  Sample random number with uniform distribution.
- static double **normal** ()
  Sample random number with normal distribution.
- static bool **bernoulli** ()
  Sample random number with Bernoulli distribution.

Static Public Attributes

- static std::mt19937 **engine**

  Mersenne Twister engine.
- static unsigned **seed** = std::mt19937::default_seed

  Seed.

5.44.1 Detailed Description

Random number generator.

Definition at line 34 of file random.hh.

5.44.2 Member Function Documentation

5.44.2.1 reset()

```
void reset ( )  [static]
```

Reset engine.

Using static member seed.

Definition at line 45 of file random.cc.

5.44.2.2 set_seed()

```
void set_seed ( )  [static]
```

Set seed.

Uses std::chrono::system_clock.

Definition at line 39 of file random.cc.

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

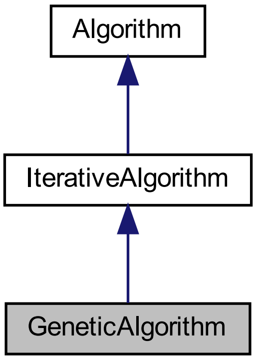
- lib/hnco/random.hh
- lib/hnco/random.cc

5.45 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



Public Member Functions

- GeneticAlgorithm (int n, int mu)

  Constructor.

Setters

- void **set_mutation_rate** (double p)

  Set the mutation rate.
- void **set_crossover_probability** (double p)

  Set the crossover probability.
- void **set_tournament_size** (int n)

  Set the tournament size.
- void **set_allow_no_mutation** (bool b)

  Set the flag _allow_no_mutation.

Protected Member Functions

Loop

- void **init** () override

  Initialize.
- void **iterate** () override

  Single iteration.

Protected Attributes

- Population **_parents**
  Parents.
- Population **_offsprings**
  Offsprings.
- CommaSelection **_comma_selection**
  Comma selection.
- TournamentSelection **_tournament_selection**
  Tournament selection.
- neighborhood::StandardBitMutation **_mutation**
  Mutation operator.
- std::bernoulli_distribution **_do_crossover**
  Do crossover.
- UniformCrossover **_crossover**
  Uniform crossover.

Parameters

- double **_mutation_rate**
  Mutation rate.
- double **_crossover_probability** = 0.5
  Crossover probability.
- int **_tournament_size** = 10
  Tournament size.
- bool **_allow_no_mutation** = false
  Allow no mutation.

5.45.1 Detailed Description

Genetic algorithm.

- Tournament selection for reproduction
- Uniform crossover
- Standard bit 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 53 of file genetic-algorithm.hh.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 GeneticAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors
mu	Population size

Definition at line 115 of file genetic-algorithm.hh.

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

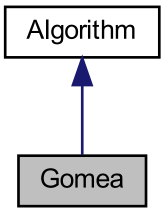
- lib/hnco/algorithms/evolutionary-algorithms/genetic-algorithm.hh
- lib/hnco/algorithms/evolutionary-algorithms/genetic-algorithm.cc

5.46 Gomea Class Reference

GOMEA.

`#include <hnco/algorithms/gomea/gomea.hh>`

Inheritance diagram for Gomea:



Public Member Functions

- **Gomea** (int n)

  Constructor.
- void **maximize** (const std::vector< function::Function ∗ > &functions)

  Maximize.
- void **finalize** ()

  Finalize.

Private Attributes

- ::gomea::linkage_config_t **_linkage_config**

  Linkage configuration.
- ::gomea::discrete::Config **_config**

  Configuration.
- std::shared_ptr< HncoFitness > **_fitness**

  Fitness.
- std::shared_ptr< hnco::function::controller::ProgressTracker > **_tracker**

  Progress tracker.

Additional Inherited Members

5.46.1 Detailed Description

GOMEA.

Implemention of the Gene-pool Optimal Mixing Evolutionary Algorithm.

Author: Anton Bouter

Integrated into HNCO by Arnaud Berny

References:

- A Joint Python/C++ Library for Efficient yet Accessible Black-Box and Gray-Box Optimization with GOMEA, Anton Bouter and Peter A.N. Bosman
- Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms, Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A. N. Bosman

Definition at line 62 of file gomea.hh.

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

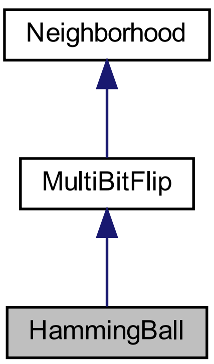
- lib/hnco/algorithms/gomea/gomea.hh
- lib/hnco/algorithms/gomea/gomea.cc

5.47 HammingBall Class Reference

Hamming ball.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for HammingBall:



Public Member Functions

- HammingBall (int n, int r)

  Constructor.

Private Member Functions

- void **sample_bits** ()

  Sample bits.

Private Attributes

- std::uniform_int_distribution< int > **_choose_k**

  Choose the distance to the center.

Additional Inherited Members

5.47.1 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 302 of file neighborhood.hh.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 HammingBall()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the ball

Definition at line 318 of file neighborhood.hh.

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

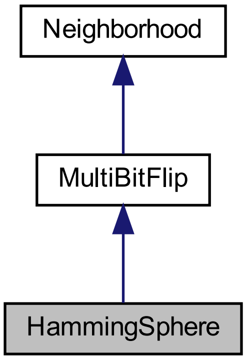
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.48 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

5.48.1 Detailed Description

Hamming sphere.

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

Definition at line 334 of file neighborhood.hh.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 HammingSphere()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the sphere

Definition at line 350 of file neighborhood.hh.

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

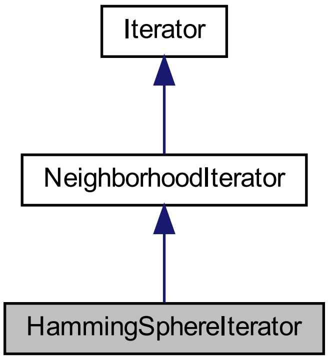
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.49 HammingSphereIterator 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** () override

    Has next bit vector.
- const bit_vector_t & **next** () override

    Next bit vector.

Private Attributes

- int **_radius**

    Radius of the ball.
- sparse_bit_vector_t **_bit_indexes**

    Bit indexes.

Additional Inherited Members

5.49.1 Detailed Description

Hamming sphere neighborhood iterator.

The Hamming sphere iterator is implemented using an array of indexes which indicate the bits to flip in the given origin.

For example, in dimension n = 4 and with radius = 2, the sequence of indexes is as follows (assuming indexes start at 1):

- 12 (first state, bits 1 and 2 are flipped)

- 13
- 14
- 23 (last index cannot be increased, first index is increased and second index is reset)
- 24
- 34

Reference: `https://en.wikipedia.org/wiki/Combination#Enumerating_k-combinations`

Definition at line 96 of file neighborhood-iterator.hh.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 HammingSphereIterator()

```
HammingSphereIterator (
            int n,
            int r )
```

Constructor.

Parameters

n	Size of bit vectors
r	Radius of Hamming Ball

Definition at line 72 of file neighborhood-iterator.cc.

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

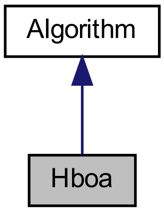
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.50 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.
- ∼**Hboa** ()
  Destructor.
- void **maximize** (const std::vector< function::Function ∗ > &functions)
  Maximize.
- void **finalize** ()
  Finalize.
- void **set_population_size** (int n)
  Set population size.

Private Attributes

- Implementation ∗ _implementation
  Pointer to implementation.
- int **_population_size** = 10
  Population size.

Additional Inherited Members

5.50.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

Implementation of the Hierarchical Bayesian Optimization Algorithm.

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Reference:

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.

Definition at line 50 of file hboa.hh.

5.50.2 Member Data Documentation

5.50.2.1 _implementation

```
Implementation* _implementation [private]
```

Pointer to implementation.

The main motivation for this pattern is to avoid including declarations from fast_efficient_p3 into the global namespace.

A raw pointer is used instead of a unique_ptr because the latter will not compile with pybind11.

Definition at line 60 of file hboa.hh.

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

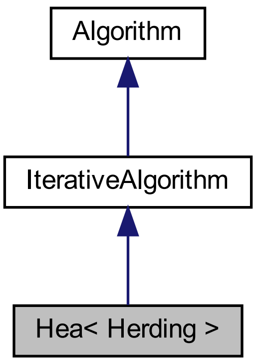
- lib/hnco/algorithms/fast-efficient-p3/hboa.hh
- lib/hnco/algorithms/fast-efficient-p3/hboa.cc

5.51 Hea< Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Herding >:



Public Member Functions

- Hea (int n, int population_size)

  Constructor.

Setters

- void **set_margin** (double x)

  Set the moment margin.
- void set_selection_size (int x)

  Set the selection size.
- void set_reset_period (int x)

  Set the reset period.
- void **set_learning_rate** (double x)

  Set the learning rate.
- void **set_bound_moment** (bool x)

  Set the bound moment after update.
- void **set_randomize_bit_order** (bool b)

  Randomize bit order.

Setters for logging

- void **set_log_herding_error** (bool b)

  Log herding error (moment discrepancy)
- void **set_log_target_norm** (bool b)

  Log target 2-norm (distance to uniform moment)
- void **set_log_delta_norm** (bool b)

  Log delta (moment increment) 2-norm.
- void **set_log_target** (bool b)

  Log target moment as a symmetric matrix.

Private Member Functions

Loop

- void **init** () override

  Initialization.
- void **iterate** () override

  Single iteration.
- void **set_something_to_log** ()

  Set flag for something to log.
- void **log** () override

  Log.

Private Attributes

- Herding::Moment **_target**

  Target moment.
- Herding::Moment **_selection**

  Moment of selected individuals.
- algorithm::Population **_population**

  Population
- Herding **_herding**

  Herding.

- double **_herding_error**

  Herding error (moment discrepancy)
- double **_target_norm**

  Target 2-norm (distance to uniform moment)
- double **_delta_norm**

  Delta (moment increment) 2-norm.

Parameters

- double **_margin**

  Moment margin.
- int **_selection_size** = 1

  Selection size.
- int **_reset_period** = 0

  Reset period.
- double **_learning_rate** = 1e-4

  Learning rate.
- bool **_bound_moment** = false

  Bound moment after update.

Logging

- bool **_log_herding_error** = false

  Log herding error (moment discrepancy)
- bool **_log_target_norm** = false

  Log target 2-norm (distance to uniform moment)
- bool **_log_delta_norm** = false

  Log delta 2-norm (moment increment)
- bool **_log_target** = false

  Log target moment as a symmetric matrix.

Additional Inherited Members

5.51.1 Detailed Description

template<class Herding>
class hnco::algorithm::walsh_moment::Hea< Herding >

Herding evolutionary algorithm.

Reference:

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

Definition at line 47 of file hea.hh.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 Hea()

```
Hea (
            int n,
            int population_size )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors
population_size	Population size

_margin is initialized to 1 / n.

Definition at line 200 of file hea.hh.

5.51.3 Member Function Documentation

5.51.3.1 set_reset_period()

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

Set the reset period.

Parameters

x	Reset period

x <= 0 means no reset.

Definition at line 229 of file hea.hh.

5.51.3.2 set_selection_size()

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

Set the selection size.

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

Definition at line 221 of file hea.hh.

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

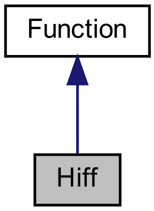
- lib/hnco/algorithms/walsh-moment/hea.hh

5.52 Hiff Class Reference

Hierarchical if and only if.

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

Inheritance diagram for Hiff:



Public Member Functions

- **Hiff** (int bv_size)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.

Private Attributes

- int **_bv_size**
  Bit vector size.
- int **_depth**
  Tree depth.

5.52.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 170 of file theory.hh.

5.52.2 Member Function Documentation

5.52.2.1 get_maximum()

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

Get the global maximum.

Returns

$(i + 1) * 2^i$ where 2^i = _bv_size

Reimplemented from Function.

Definition at line 195 of file theory.hh.

5.52.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 191 of file theory.hh.

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

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

5.53 HncoEvaluator Class Reference

Evaluator for HNCO functions.

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

Inherits Evaluator.

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.

5.53.1 Detailed Description

Evaluator for HNCO functions.

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

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

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

5.54 HncoFitness Class Reference

Fitness for HNCO functions.

```
#include <hnco/algorithms/gomea/hnco-fitness.hh>
```

Inherits BBOFitnessFunction_t< char >.

Public Member Functions

- **HncoFitness** (hnco::function::Function ∗function)
  Constructor.
- double **objectiveFunction** (int objective_index, ::gomea::vec_t< char > &variables) override
  Evaluate a bit vector.

Private Attributes

- hnco::function::Function ∗ **_function**
  HNCO function.
- hnco::bit_vector_t **_bv**
  Argument of HNCO function.

5.54.1 Detailed Description

Fitness for HNCO functions.

Definition at line 35 of file hnco-fitness.hh.

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

- lib/hnco/algorithms/gomea/hnco-fitness.hh

5.55 HncoOptions Class Reference

Command line options for hnco.

`#include <hnco/app/hnco-options.hh>`

Public Member Functions

- **HncoOptions** ()
  Default constructor.
- **HncoOptions** (int argc, char *argv[ ], bool ignore_bad_options=false)
  Constructor.
- int **get_algorithm** () const
  Get the value of algorithm.
- bool **with_algorithm** () const
  With parameter algorithm.
- int **get_bm_num_gs_cycles** () const
  Get the value of bm_num_gs_cycles.
- bool **with_bm_num_gs_cycles** () const
  With parameter bm_num_gs_cycles.
- int **get_bm_num_gs_steps** () const
  Get the value of bm_num_gs_steps.
- bool **with_bm_num_gs_steps** () const
  With parameter bm_num_gs_steps.
- int **get_bm_reset_mode** () const
  Get the value of bm_reset_mode.
- bool **with_bm_reset_mode** () const
  With parameter bm_reset_mode.
- int **get_bm_sampling_mode** () const
  Get the value of bm_sampling_mode.
- bool **with_bm_sampling_mode** () const
  With parameter bm_sampling_mode.
- int **get_budget** () const
  Get the value of budget.
- bool **with_budget** () const
  With parameter budget.
- int **get_bv_size** () const
  Get the value of bv_size.
- bool **with_bv_size** () const

With parameter bv_size.

- std::string **get_description_path** () const
  Get the value of description_path.
- bool **with_description_path** () const
  With parameter description_path.
- double **get_ea_crossover_bias** () const
  Get the value of ea_crossover_bias.
- bool **with_ea_crossover_bias** () const
  With parameter ea_crossover_bias.
- double **get_ea_crossover_probability** () const
  Get the value of ea_crossover_probability.
- bool **with_ea_crossover_probability** () const
  With parameter ea_crossover_probability.
- int **get_ea_it_initial_hamming_weight** () const
  Get the value of ea_it_initial_hamming_weight.
- bool **with_ea_it_initial_hamming_weight** () const
  With parameter ea_it_initial_hamming_weight.
- int **get_ea_it_replacement** () const
  Get the value of ea_it_replacement.
- bool **with_ea_it_replacement** () const
  With parameter ea_it_replacement.
- int **get_ea_lambda** () const
  Get the value of ea_lambda.
- bool **with_ea_lambda** () const
  With parameter ea_lambda.
- int **get_ea_mu** () const
  Get the value of ea_mu.
- bool **with_ea_mu** () const
  With parameter ea_mu.
- double **get_ea_mutation_rate** () const
  Get the value of ea_mutation_rate.
- bool **with_ea_mutation_rate** () const
  With parameter ea_mutation_rate.
- double **get_ea_mutation_rate_max** () const
  Get the value of ea_mutation_rate_max.
- bool **with_ea_mutation_rate_max** () const
  With parameter ea_mutation_rate_max.
- double **get_ea_mutation_rate_min** () const
  Get the value of ea_mutation_rate_min.
- bool **with_ea_mutation_rate_min** () const
  With parameter ea_mutation_rate_min.
- double **get_ea_success_ratio** () const
  Get the value of ea_success_ratio.
- bool **with_ea_success_ratio** () const
  With parameter ea_success_ratio.
- int **get_ea_tournament_size** () const
  Get the value of ea_tournament_size.
- bool **with_ea_tournament_size** () const
  With parameter ea_tournament_size.
- double **get_ea_update_strength** () const
  Get the value of ea_update_strength.

- bool **with_ea_update_strength** () const
  With parameter ea_update_strength.
- std::string **get_expression** () const
  Get the value of expression.
- bool **with_expression** () const
  With parameter expression.
- std::string **get_fn_name** () const
  Get the value of fn_name.
- bool **with_fn_name** () const
  With parameter fn_name.
- int **get_fn_num_traps** () const
  Get the value of fn_num_traps.
- bool **with_fn_num_traps** () const
  With parameter fn_num_traps.
- int **get_fn_prefix_length** () const
  Get the value of fn_prefix_length.
- bool **with_fn_prefix_length** () const
  With parameter fn_prefix_length.
- int **get_fn_threshold** () const
  Get the value of fn_threshold.
- bool **with_fn_threshold** () const
  With parameter fn_threshold.
- double **get_fp_default_double_precision** () const
  Get the value of fp_default_double_precision.
- bool **with_fp_default_double_precision** () const
  With parameter fp_default_double_precision.
- std::string **get_fp_default_double_rep** () const
  Get the value of fp_default_double_rep.
- bool **with_fp_default_double_rep** () const
  With parameter fp_default_double_rep.
- int **get_fp_default_double_size** () const
  Get the value of fp_default_double_size.
- bool **with_fp_default_double_size** () const
  With parameter fp_default_double_size.
- std::string **get_fp_default_int_rep** () const
  Get the value of fp_default_int_rep.
- bool **with_fp_default_int_rep** () const
  With parameter fp_default_int_rep.
- std::string **get_fp_default_long_rep** () const
  Get the value of fp_default_long_rep.
- bool **with_fp_default_long_rep** () const
  With parameter fp_default_long_rep.
- std::string **get_fp_expression** () const
  Get the value of fp_expression.
- bool **with_fp_expression** () const
  With parameter fp_expression.
- int **get_fp_expression_source** () const
  Get the value of fp_expression_source.
- bool **with_fp_expression_source** () const
  With parameter fp_expression_source.
- std::string **get_fp_representations** () const

Get the value of fp_representations.
- bool **with_fp_representations** () const

  With parameter fp_representations.
- std::string **get_fp_representations_path** () const

  Get the value of fp_representations_path.
- bool **with_fp_representations_path** () const

  With parameter fp_representations_path.
- int **get_fp_representations_source** () const

  Get the value of fp_representations_source.
- bool **with_fp_representations_source** () const

  With parameter fp_representations_source.
- int **get_function** () const

  Get the value of function.
- bool **with_function** () const

  With parameter function.
- int **get_hea_reset_period** () const

  Get the value of hea_reset_period.
- bool **with_hea_reset_period** () const

  With parameter hea_reset_period.
- double **get_learning_rate** () const

  Get the value of learning_rate.
- bool **with_learning_rate** () const

  With parameter learning_rate.
- int **get_map** () const

  Get the value of map.
- bool **with_map** () const

  With parameter map.
- int **get_map_input_size** () const

  Get the value of map_input_size.
- bool **with_map_input_size** () const

  With parameter map_input_size.
- std::string **get_map_path** () const

  Get the value of map_path.
- bool **with_map_path** () const

  With parameter map_path.
- int **get_map_ts_length** () const

  Get the value of map_ts_length.
- bool **with_map_ts_length** () const

  With parameter map_ts_length.
- int **get_map_ts_sampling_mode** () const

  Get the value of map_ts_sampling_mode.
- bool **with_map_ts_sampling_mode** () const

  With parameter map_ts_sampling_mode.
- int **get_neighborhood** () const

  Get the value of neighborhood.
- bool **with_neighborhood** () const

  With parameter neighborhood.
- int **get_neighborhood_iterator** () const

  Get the value of neighborhood_iterator.
- bool **with_neighborhood_iterator** () const

  With parameter neighborhood_iterator.

- double **get_noise_stddev** () const

  Get the value of noise_stddev.
- bool **with_noise_stddev** () const

  With parameter noise_stddev.
- int **get_num_iterations** () const

  Get the value of num_iterations.
- bool **with_num_iterations** () const

  With parameter num_iterations.
- int **get_num_threads** () const

  Get the value of num_threads.
- bool **with_num_threads** () const

  With parameter num_threads.
- std::string **get_path** () const

  Get the value of path.
- bool **with_path** () const

  With parameter path.
- double **get_pn_mutation_rate** () const

  Get the value of pn_mutation_rate.
- bool **with_pn_mutation_rate** () const

  With parameter pn_mutation_rate.
- int **get_pn_neighborhood** () const

  Get the value of pn_neighborhood.
- bool **with_pn_neighborhood** () const

  With parameter pn_neighborhood.
- int **get_pn_radius** () const

  Get the value of pn_radius.
- bool **with_pn_radius** () const

  With parameter pn_radius.
- int **get_population_size** () const

  Get the value of population_size.
- bool **with_population_size** () const

  With parameter population_size.
- int **get_pv_log_num_components** () const

  Get the value of pv_log_num_components.
- bool **with_pv_log_num_components** () const

  With parameter pv_log_num_components.
- int **get_radius** () const

  Get the value of radius.
- bool **with_radius** () const

  With parameter radius.
- int **get_rep_categorical_representation** () const

  Get the value of rep_categorical_representation.
- bool **with_rep_categorical_representation** () const

  With parameter rep_categorical_representation.
- int **get_rep_num_additional_bits** () const

  Get the value of rep_num_additional_bits.
- bool **with_rep_num_additional_bits** () const

  With parameter rep_num_additional_bits.
- std::string **get_results_path** () const

  Get the value of results_path.
- bool **with_results_path** () const

With parameter results_path.

- int **get_rls_patience** () const

  Get the value of rls_patience.
- bool **with_rls_patience** () const

  With parameter rls_patience.
- double **get_sa_beta_ratio** () const

  Get the value of sa_beta_ratio.
- bool **with_sa_beta_ratio** () const

  With parameter sa_beta_ratio.
- double **get_sa_initial_acceptance_probability** () const

  Get the value of sa_initial_acceptance_probability.
- bool **with_sa_initial_acceptance_probability** () const

  With parameter sa_initial_acceptance_probability.
- int **get_sa_num_transitions** () const

  Get the value of sa_num_transitions.
- bool **with_sa_num_transitions** () const

  With parameter sa_num_transitions.
- int **get_sa_num_trials** () const

  Get the value of sa_num_trials.
- bool **with_sa_num_trials** () const

  With parameter sa_num_trials.
- unsigned **get_seed** () const

  Get the value of seed.
- bool **with_seed** () const

  With parameter seed.
- int **get_selection_size** () const

  Get the value of selection_size.
- bool **with_selection_size** () const

  With parameter selection_size.
- std::string **get_solution_path** () const

  Get the value of solution_path.
- bool **with_solution_path** () const

  With parameter solution_path.
- double **get_target** () const

  Get the value of target.
- bool **with_target** () const

  With parameter target.
- bool **with_additive_gaussian_noise** () const

  With the flag additive_gaussian_noise.
- bool **with_bm_log_norm_1** () const

  With the flag bm_log_norm_1.
- bool **with_bm_log_norm_infinite** () const

  With the flag bm_log_norm_infinite.
- bool **with_bm_negative_positive_selection** () const

  With the flag bm_negative_positive_selection.
- bool **with_cache** () const

  With the flag cache.
- bool **with_cache_budget** () const

  With the flag cache_budget.
- bool **with_concrete_solution** () const

  With the flag concrete_solution.

- bool **with_ea_allow_no_mutation** () const
  With the flag ea_allow_no_mutation.
- bool **with_ea_it_log_center_fitness** () const
  With the flag ea_it_log_center_fitness.
- bool **with_ea_log_mutation_rate** () const
  With the flag ea_log_mutation_rate.
- bool **with_fn_display** () const
  With the flag fn_display.
- bool **with_fn_get_bv_size** () const
  With the flag fn_get_bv_size.
- bool **with_fn_get_maximum** () const
  With the flag fn_get_maximum.
- bool **with_fn_has_known_maximum** () const
  With the flag fn_has_known_maximum.
- bool **with_fn_provides_incremental_evaluation** () const
  With the flag fn_provides_incremental_evaluation.
- bool **with_fn_walsh_transform** () const
  With the flag fn_walsh_transform.
- bool **with_hea_bound_moment** () const
  With the flag hea_bound_moment.
- bool **with_hea_log_delta_norm** () const
  With the flag hea_log_delta_norm.
- bool **with_hea_log_herding_error** () const
  With the flag hea_log_herding_error.
- bool **with_hea_log_target** () const
  With the flag hea_log_target.
- bool **with_hea_log_target_norm** () const
  With the flag hea_log_target_norm.
- bool **with_hea_randomize_bit_order** () const
  With the flag hea_randomize_bit_order.
- bool **with_incremental_evaluation** () const
  With the flag incremental_evaluation.
- bool **with_load_solution** () const
  With the flag load_solution.
- bool **with_log_improvement** () const
  With the flag log_improvement.
- bool **with_map_display** () const
  With the flag map_display.
- bool **with_map_random** () const
  With the flag map_random.
- bool **with_map_surjective** () const
  With the flag map_surjective.
- bool **with_mmas_strict** () const
  With the flag mmas_strict.
- bool **with_negation** () const
  With the flag negation.
- bool **with_parsed_modifier** () const
  With the flag parsed_modifier.
- bool **with_pn_allow_no_mutation** () const
  With the flag pn_allow_no_mutation.
- bool **with_print_default_parameters** () const

With the flag print_default_parameters.
- bool **with_print_description** () const
  With the flag print_description.
- bool **with_print_parameters** () const
  With the flag print_parameters.
- bool **with_print_results** () const
  With the flag print_results.
- bool **with_print_solution** () const
  With the flag print_solution.
- bool **with_prior_noise** () const
  With the flag prior_noise.
- bool **with_pv_log_entropy** () const
  With the flag pv_log_entropy.
- bool **with_pv_log_pv** () const
  With the flag pv_log_pv.
- bool **with_record_evaluation_time** () const
  With the flag record_evaluation_time.
- bool **with_record_total_time** () const
  With the flag record_total_time.
- bool **with_restart** () const
  With the flag restart.
- bool **with_rls_strict** () const
  With the flag rls_strict.
- bool **with_rw_log_value** () const
  With the flag rw_log_value.
- bool **with_save_description** () const
  With the flag save_description.
- bool **with_save_results** () const
  With the flag save_results.
- bool **with_save_solution** () const
  With the flag save_solution.
- bool **with_stop_on_maximum** () const
  With the flag stop_on_maximum.
- bool **with_stop_on_target** () const
  With the flag stop_on_target.

Private Member Functions

- void **print_help** (std::ostream &stream) const
  Print help message.
- void **print_help_fn** (std::ostream &stream) const
  Print help message for section fn.
- void **print_help_fp** (std::ostream &stream) const
  Print help message for section fp.
- void **print_help_rep** (std::ostream &stream) const
  Print help message for section rep.
- void **print_help_mod** (std::ostream &stream) const
  Print help message for section mod.
- void **print_help_ctrl** (std::ostream &stream) const
  Print help message for section ctrl.

- void **print_help_pn** (std::ostream &stream) const
  Print help message for section pn.
- void **print_help_map** (std::ostream &stream) const
  Print help message for section map.
- void **print_help_alg** (std::ostream &stream) const
  Print help message for section alg.
- void **print_help_ls** (std::ostream &stream) const
  Print help message for section ls.
- void **print_help_sa** (std::ostream &stream) const
  Print help message for section sa.
- void **print_help_ea** (std::ostream &stream) const
  Print help message for section ea.
- void **print_help_eda** (std::ostream &stream) const
  Print help message for section eda.
- void **print_help_hea** (std::ostream &stream) const
  Print help message for section hea.
- void **print_help_bm** (std::ostream &stream) const
  Print help message for section bm.
- void **print_version** (std::ostream &stream) const
  Print version.

Private Attributes

- std::string **_exec_name**
  Name of the executable.
- std::string **_version** = "0.24"
  Name Version.
- int **_algorithm** = 100
  Type of algorithm.
- int **_bm_num_gs_cycles** = 1
  Number of Gibbs sampler cycles per bit vector.
- int **_bm_num_gs_steps** = 100
  Number of Gibbs sampler steps per bit vector.
- int **_bm_reset_mode** = 1
  Markov chain reset mode.
- int **_bm_sampling_mode** = 1
  Sampling mode for the Boltzmann machine.
- int **_budget** = 10000
  Number of allowed function evaluations (<= 0 means indefinite)
- int **_bv_size** = 100
  Size of bit vectors.
- std::string **_description_path** = "description.txt"
  Path of the description file.
- double **_ea_crossover_bias** = 0.5
  Crossover bias.
- double **_ea_crossover_probability** = 0.5
  Crossover probability.
- int **_ea_it_initial_hamming_weight** = 0
  Initial Hamming weight.
- int **_ea_it_replacement** = 0

Selection for replacement in it-EA.
- int **_ea_lambda** = 100

  Offspring population size.
- int **_ea_mu** = 10

  Parent population size.
- double **_ea_mutation_rate**

  Mutation rate (fixed or initial value)
- double **_ea_mutation_rate_max** = 0.5

  Maximum mutation rate.
- double **_ea_mutation_rate_min**

  Minimum mutation rate.
- double **_ea_success_ratio** = 4

  Success rate for for self-adjusting mutation rate.
- int **_ea_tournament_size** = 2

  Tournament size.
- double **_ea_update_strength** = 1.01

  Update strength for self-adjusting mutation rate.
- std::string **_expression** = "x"

  Expression of the variable x.
- std::string **_fn_name**

  Name of the function in the dynamic library.
- int **_fn_num_traps** = 10

  Number of traps.
- int **_fn_prefix_length** = 2

  Prefix length for long path.
- int **_fn_threshold** = 10

  Threshold (in bits) for Jump, Four Peaks, and Six Peaks.
- double **_fp_default_double_precision**

  Default precision of double representations.
- std::string **_fp_default_double_rep** = "double(0, 1, precision = 1e-3)"

  Default representation for double.
- int **_fp_default_double_size**

  Default size of double representations.
- std::string **_fp_default_int_rep** = "int(1, 100)"

  Default representation for int.
- std::string **_fp_default_long_rep** = "long(1, 100)"

  Default representation for long.
- std::string **_fp_expression** = "(1-x)^2+100∗(y-x^2)^2"

  Mathematical expression.
- int **_fp_expression_source** = 0

  Source for the expression to parse.
- std::string **_fp_representations**

  Representations. Example: "x: double(0, 1); y: double(0, 1, precision = 1e-3); z: double(0, 1, size = 8); u: int(-100, 100); v: long(1, 10000)".
- std::string **_fp_representations_path** = "representations.txt"

  Path of the representations file.
- int **_fp_representations_source** = 0

  Source for the representations.
- int **_function** = 0

  Type of function.
- int **_hea_reset_period** = 0

Reset period (<= 0 means no reset)

- double **_learning_rate** = 0.001

  Learning rate.
- int **_map** = 0

  Type of map.
- int **_map_input_size** = 100

  Input size of linear and affine maps.
- std::string **_map_path** = "map.txt"

  Path of the map file.
- int **_map_ts_length** = 10

  Transvection sequence length.
- int **_map_ts_sampling_mode** = 0

  Transvection sequence sampling mode.
- int **_neighborhood** = 0

  Type of neighborhood.
- int **_neighborhood_iterator** = 0

  Type of neighborhood iterator.
- double **_noise_stddev** = 1

  Noise standard deviation.
- int **_num_iterations** = 0

  Number of iterations (<= 0 means indefinite)
- int **_num_threads** = 1

  Number of threads.
- std::string **_path** = "function.txt"

  Path of the function file.
- double **_pn_mutation_rate**

  Mutation rate.
- int **_pn_neighborhood** = 0

  Type of neighborhood.
- int **_pn_radius** = 2

  Radius of Hamming ball or sphere.
- int **_population_size** = 10

  Population size.
- int **_pv_log_num_components** = 5

  Number of probability vector components to log.
- int **_radius** = 2

  Radius of Hamming ball or sphere.
- int **_rep_categorical_representation** = 0

  Categorical representation.
- int **_rep_num_additional_bits** = 2

  Number of additional bits per element for permutation representation.
- std::string **_results_path** = "results.json"

  Path of the results file.
- int **_rls_patience** = 50

  Number of consecutive rejected moves before ending the search (<= 0 means infinite)
- double **_sa_beta_ratio** = 1.2

  Ratio for beta or inverse temperature.
- double **_sa_initial_acceptance_probability** = 0.6

  Initial acceptance probability.
- int **_sa_num_transitions** = 50

  Number of accepted transitions before annealing.

- int **_sa_num_trials** = 100
  Number of trials to estimate initial inverse temperature.
- unsigned **_seed**
  Seed for the random number generator.
- int **_selection_size** = 1
  Selection size (number of selected individuals)
- std::string **_solution_path** = "solution.txt"
  Path of the solution file.
- double **_target** = 100
  Target.
- bool **_additive_gaussian_noise** = false
  Additive Gaussian noise.
- bool **_bm_log_norm_1** = false
  Log 1-norm of the parameters.
- bool **_bm_log_norm_infinite** = false
  Log infinite norm of the parameters.
- bool **_bm_negative_positive_selection** = false
  Negative and positive selection.
- bool **_cache** = false
  Cache function evaluations.
- bool **_cache_budget** = false
  Set cache on budget.
- bool **_concrete_solution** = false
  Print or save the solution in the domain of the concrete function.
- bool **_ea_allow_no_mutation** = false
  Allow no mutation with standard bit mutation.
- bool **_ea_it_log_center_fitness** = false
  Log center fitness.
- bool **_ea_log_mutation_rate** = false
  Log mutation rate.
- bool **_fn_display** = false
  Display the function and exit.
- bool **_fn_get_bv_size** = false
  Print the size of bit vectors.
- bool **_fn_get_maximum** = false
  If the maximum is known then print it and exit with status 0 else exit with status 1.
- bool **_fn_has_known_maximum** = false
  Check whether the function has a known maximum.
- bool **_fn_provides_incremental_evaluation** = false
  Check whether the function provides incremental evaluation.
- bool **_fn_walsh_transform** = false
  Compute the Walsh transform of the function.
- bool **_hea_bound_moment** = false
  Bound moment after update.
- bool **_hea_log_delta_norm** = false
  Log delta (moment increment) 2-norm.
- bool **_hea_log_herding_error** = false
  Log herding error (moment discrepancy)
- bool **_hea_log_target** = false
  Log target moment as a symmetric matrix.
- bool **_hea_log_target_norm** = false

Log target 2-norm (distance to uniform moment)

- bool **_hea_randomize_bit_order** = false

  Randomize bit order.
- bool **_incremental_evaluation** = false

  Incremental evaluation.
- bool **_load_solution** = false

  Load a solution from a file.
- bool **_log_improvement** = false

  Log improvement.
- bool **_map_display** = false

  Display the map and exit.
- bool **_map_random** = false

  Sample a random map.
- bool **_map_surjective** = false

  Ensure that the sampled linear or affine map is surjective.
- bool **_mmas_strict** = false

  Strict (>) max-min ant system.
- bool **_negation** = false

  Negation (hence minimization) of the function.
- bool **_parsed_modifier** = false

  Parsed modifier.
- bool **_pn_allow_no_mutation** = false

  Allow no mutation with standard bit mutation.
- bool **_print_default_parameters** = false

  Print the default parameters and exit.
- bool **_print_description** = false

  Print a description of the solution.
- bool **_print_parameters** = false

  Print the parameters.
- bool **_print_results** = false

  Print results.
- bool **_print_solution** = false

  Print the solution.
- bool **_prior_noise** = false

  Prior noise.
- bool **_pv_log_entropy** = false

  Log entropy of probability vector.
- bool **_pv_log_pv** = false

  Log probability vector.
- bool **_record_evaluation_time** = false

  Record evaluation time.
- bool **_record_total_time** = false

  Record total time.
- bool **_restart** = false

  Restart any algorithm an indefinite number of times.
- bool **_rls_strict** = false

  Strict (>) random local search.
- bool **_rw_log_value** = false

  Log bit vector value during random walk.
- bool **_save_description** = false

  Save the description of the solution in a file.

- bool **_save_results** = false

  Save the results in a file.
- bool **_save_solution** = false

  Save the solution in a file.
- bool **_stop_on_maximum** = false

  Stop on maximum.
- bool **_stop_on_target** = false

  Stop on target.

Friends

- std::ostream & **operator**<< (std::ostream &, const HncoOptions &)

  Print a header containing the parameter values.

5.55.1 Detailed Description

Command line options for hnco.

Definition at line 11 of file hnco-options.hh.

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

- lib/hnco/app/hnco-options.hh
- lib/hnco/app/hnco-options.cc

5.56 HncoOptions Class Reference

Command line options for hnco-mo.

```
#include <hnco/multiobjective/app/hnco-mo-options.hh>
```

Public Member Functions

- **HncoOptions** ()

  Default constructor.
- **HncoOptions** (int argc, char *argv[ ], bool ignore_bad_options=false)

  Constructor.
- int **get_algorithm** () const

  Get the value of algorithm.
- bool **with_algorithm** () const

  With parameter algorithm.
- int **get_bv_size** () const

  Get the value of bv_size.
- bool **with_bv_size** () const

  With parameter bv_size.
- double **get_ea_crossover_probability** () const

  Get the value of ea_crossover_probability.

- bool **with_ea_crossover_probability** () const
  With parameter ea_crossover_probability.
- int **get_ea_mu** () const
  Get the value of ea_mu.
- bool **with_ea_mu** () const
  With parameter ea_mu.
- double **get_ea_mutation_rate** () const
  Get the value of ea_mutation_rate.
- bool **with_ea_mutation_rate** () const
  With parameter ea_mutation_rate.
- int **get_ea_tournament_size** () const
  Get the value of ea_tournament_size.
- bool **with_ea_tournament_size** () const
  With parameter ea_tournament_size.
- std::string **get_fn_name** () const
  Get the value of fn_name.
- bool **with_fn_name** () const
  With parameter fn_name.
- double **get_fp_default_double_precision** () const
  Get the value of fp_default_double_precision.
- bool **with_fp_default_double_precision** () const
  With parameter fp_default_double_precision.
- std::string **get_fp_default_double_rep** () const
  Get the value of fp_default_double_rep.
- bool **with_fp_default_double_rep** () const
  With parameter fp_default_double_rep.
- int **get_fp_default_double_size** () const
  Get the value of fp_default_double_size.
- bool **with_fp_default_double_size** () const
  With parameter fp_default_double_size.
- std::string **get_fp_default_int_rep** () const
  Get the value of fp_default_int_rep.
- bool **with_fp_default_int_rep** () const
  With parameter fp_default_int_rep.
- std::string **get_fp_default_long_rep** () const
  Get the value of fp_default_long_rep.
- bool **with_fp_default_long_rep** () const
  With parameter fp_default_long_rep.
- std::string **get_fp_expression** () const
  Get the value of fp_expression.
- bool **with_fp_expression** () const
  With parameter fp_expression.
- int **get_fp_expression_source** () const
  Get the value of fp_expression_source.
- bool **with_fp_expression_source** () const
  With parameter fp_expression_source.
- std::string **get_fp_representations** () const
  Get the value of fp_representations.
- bool **with_fp_representations** () const
  With parameter fp_representations.
- std::string **get_fp_representations_path** () const

Get the value of fp_representations_path.
- bool **with_fp_representations_path** () const
  With parameter fp_representations_path.
- int **get_fp_representations_source** () const
  Get the value of fp_representations_source.
- bool **with_fp_representations_source** () const
  With parameter fp_representations_source.
- int **get_function** () const
  Get the value of function.
- bool **with_function** () const
  With parameter function.
- int **get_num_iterations** () const
  Get the value of num_iterations.
- bool **with_num_iterations** () const
  With parameter num_iterations.
- int **get_num_threads** () const
  Get the value of num_threads.
- bool **with_num_threads** () const
  With parameter num_threads.
- std::string **get_path** () const
  Get the value of path.
- bool **with_path** () const
  With parameter path.
- int **get_rep_categorical_representation** () const
  Get the value of rep_categorical_representation.
- bool **with_rep_categorical_representation** () const
  With parameter rep_categorical_representation.
- int **get_rep_num_additional_bits** () const
  Get the value of rep_num_additional_bits.
- bool **with_rep_num_additional_bits** () const
  With parameter rep_num_additional_bits.
- unsigned **get_seed** () const
  Get the value of seed.
- bool **with_seed** () const
  With parameter seed.
- bool **with_ea_allow_no_mutation** () const
  With the flag ea_allow_no_mutation.
- bool **with_fn_display** () const
  With the flag fn_display.
- bool **with_fn_get_bv_size** () const
  With the flag fn_get_bv_size.
- bool **with_fn_get_output_size** () const
  With the flag fn_get_output_size.
- bool **with_print_default_parameters** () const
  With the flag print_default_parameters.
- bool **with_print_description** () const
  With the flag print_description.
- bool **with_print_parameters** () const
  With the flag print_parameters.
- bool **with_print_pareto_front** () const
  With the flag print_pareto_front.

Private Member Functions

- void **print_help** (std::ostream &stream) const
  Print help message.
- void **print_help_fn** (std::ostream &stream) const
  Print help message for section fn.
- void **print_help_fp** (std::ostream &stream) const
  Print help message for section fp.
- void **print_help_rep** (std::ostream &stream) const
  Print help message for section rep.
- void **print_help_alg** (std::ostream &stream) const
  Print help message for section alg.
- void **print_help_ea** (std::ostream &stream) const
  Print help message for section ea.
- void **print_version** (std::ostream &stream) const
  Print version.

Private Attributes

- std::string **_exec_name**
  Name of the executable.
- std::string **_version** = "0.24"
  Name Version.
- int **_algorithm** = 100
  Type of algorithm.
- int **_bv_size** = 100
  Size of bit vectors.
- double **_ea_crossover_probability** = 0.8
  Crossover probability.
- int **_ea_mu** = 100
  Parent population size.
- double **_ea_mutation_rate** = 1
  Mutation rate relative to bv_size.
- int **_ea_tournament_size** = 2
  Tournament size.
- std::string **_fn_name**
  Name of the function in the dynamic library.
- double **_fp_default_double_precision**
  Default precision of double representations.
- std::string **_fp_default_double_rep** = "double(0, 1, precision = 1e-3)"
  Default representation for double.
- int **_fp_default_double_size**
  Default size of double representations.
- std::string **_fp_default_int_rep** = "int(1, 100)"
  Default representation for int.
- std::string **_fp_default_long_rep** = "long(1, 100)"
  Default representation for long.
- std::string **_fp_expression** = "A := sin(x) + cos(y); A :: B := sqrt(x^2 + y^2); B"
  Mathematical expression (list of objectives separated by ::)
- int **_fp_expression_source** = 0

Source for the expression to parse.
- std::string **_fp_representations**

  Representations. Example: "x: double(0, 1); y: double(0, 1, precision = 1e-3); z: double(0, 1, size = 8); u: int(-100, 100); v: long(1, 10000)".
- std::string **_fp_representations_path** = "representations.txt"

  Path of the representations file.
- int **_fp_representations_source** = 0

  Source for the representations.
- int **_function** = 180

  Type of function.
- int **_num_iterations** = 100

  Number of iterations.
- int **_num_threads** = 1

  Number of threads.
- std::string **_path** = "function.txt"

  Path of a function file.
- int **_rep_categorical_representation** = 0

  Categorical representation.
- int **_rep_num_additional_bits** = 2

  Number of additional bits per element for permutation representation.
- unsigned **_seed**

  Seed for the random number generator.
- bool **_ea_allow_no_mutation** = false

  Allow no mutation with standard bit mutation.
- bool **_fn_display** = false

  Display the function and exit.
- bool **_fn_get_bv_size** = false

  Print the size of bit vectors.
- bool **_fn_get_output_size** = false

  Print the number of objectives.
- bool **_print_default_parameters** = false

  Print the parameters and exit.
- bool **_print_description** = false

  Print a description of the solution.
- bool **_print_parameters** = false

  Print the parameters.
- bool **_print_pareto_front** = false

  Print the Pareto front.

Friends

- std::ostream & **operator**<< (std::ostream &, const HncoOptions &)

  Print a header containing the parameter values.

5.56.1 Detailed Description

Command line options for hnco-mo.

Definition at line 12 of file hnco-mo-options.hh.

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

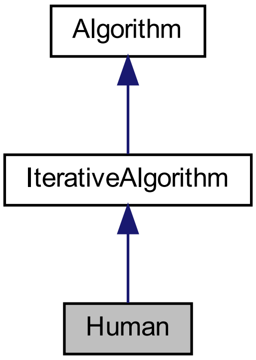
- lib/hnco/multiobjective/app/hnco-mo-options.hh
- lib/hnco/multiobjective/app/hnco-mo-options.cc

5.57 Human Class Reference

Human

`#include <hnco/algorithms/human.hh>`

Inheritance diagram for Human:



Public Member Functions

- **Human** (int n)

    Constructor.

Protected Member Functions

- void **parse_bit_vector** ()

    Parse bit vector.

Loop

- void **init** () override

    Initialize.
- void **iterate** () override

    Single iteration.

Protected Attributes

- bit_vector_t **_candidate**

    Candidate.

5.57.1 Detailed Description

Human

Definition at line 32 of file human.hh.

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

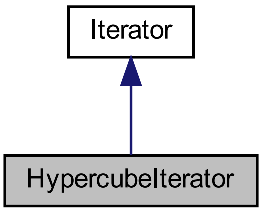
- lib/hnco/algorithms/human.hh
- lib/hnco/algorithms/human.cc

5.58 HypercubeIterator Class Reference

Hypercube iterator.

#include <hnco/iterator.hh>

Inheritance diagram for HypercubeIterator:



Public Member Functions

- **HypercubeIterator** (int n)

  Constructor.
- bool **has_next** () override

  Has next bit vector.
- const bit_vector_t & **next** () override

  Next bit vector.

Additional Inherited Members

5.58.1 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

5.59 Implementation Struct Reference

Implementation

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

Public Attributes

- Configuration **configuration**

  Configuration.
- std::shared_ptr< HncoEvaluator > **evaluator**

  Evaluator.
- std::shared_ptr< Middle_Layer > **middle_layer**

  Middle layer.

5.59.1 Detailed Description

Implementation

Definition at line 37 of file implementation.hh.

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

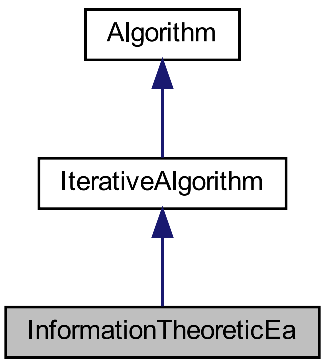
- lib/hnco/algorithms/fast-efficient-p3/implementation.hh

5.60 InformationTheoreticEa Class Reference

Information-theoretic evolutionary algorithm.

`#include <hnco/algorithms/evolutionary-algorithms/it-ea.hh>`

Inheritance diagram for InformationTheoreticEa:



Public Types

- enum class Replacement {
  elitist = 0 , non_elitist = 1 , ml_update = 2 , incremental_ml_update = 3 ,
  no_replacement = 4 }

  Selection for replacement.

Public Member Functions

- **InformationTheoreticEa** (int n, int population_size)

  Constructor.

Setters

- void **set_selection_size** (int n)
  Set the selection size.
- void **set_learning_rate** (double r)
  Set the learning rate.
- void **set_mutation_rate_init** (double r)
  Set the initial mutation rate.
- void **set_mutation_rate_min** (double r)
  Set the minimum mutation rate.
- void **set_mutation_rate_max** (double r)
  Set the maximum mutation rate.
- void **set_replacement** (Replacement replacement)
  Set replacement.
- void **set_initial_hamming_weight** (int n)
  Set the initial Hamming weight.
- void **set_allow_no_mutation** (bool b)
  Allow no mutation.

Setters for logging

- void **set_log_mutation_rate** (bool b)
  Log mutation rate.
- void **set_log_center_fitness** (bool b)
  Log center fitness.

Protected Member Functions

- void **set_something_to_log** ()

  Set flag for something to log.
- void **compute_masks** (bool equivalent_individuals, std::pair< int, int > range, double c)

  Compute masks.
- void **ml_update** (bool equivalent_individuals, std::pair< int, int > range, double c)

  ML update.
- void **incremental_ml_update** (bool equivalent_individuals, std::pair< int, int > range, double c)

  Incremental ML update.
- void **igo_update** (bool equivalent_individuals, std::pair< int, int > range, double c)

  IGO update.

Loop

- void **init** () override
  Initialization.
- void **iterate** () override
  Single iteration.
- void **log** () override
  Log.

Protected Attributes

- Population **_population**

    Population
- std::vector< bit_vector_t > **_masks**

    Mutation masks.
- std::vector< double > **_likelihoods**

    Mutation likelihoods.
- neighborhood::StandardBitMutation **_mutation_operator**

    Mutation operator.
- solution_t **_center**

    Center of the search distribution.
- double **_mutation_rate**

    Mutation rate.

Parameters

- int **_selection_size** = 1

    Selection size.
- double **_learning_rate** = 0.01

    Learning rate.
- double **_mutation_rate_init**

    Initial mutation rate.
- double **_mutation_rate_min**

    Minimum mutation rate.
- double **_mutation_rate_max** = 0.5

    Maximum mutation rate.
- int **_initial_hamming_weight** = 0

    Initial Hamming weight.
- Replacement **_replacement** = Replacement::elitist

    Replacement.
- bool **_allow_no_mutation** = false

    Allow no mutation.

Logging

- bool **_log_mutation_rate** = false

    Log entropy.
- bool **_log_center_fitness** = false

    Log center fitness.

5.60.1 Detailed Description

Information-theoretic evolutionary algorithm.

Definition at line 18 of file it-ea.hh.

5.60.2 Member Enumeration Documentation

5.60.2.1 Replacement

```
enum class Replacement  [strong]
```

Selection for replacement.

Enumerator

elitist	Elitist replacement.
non_elitist	Non elitist replacement.
ml_update	Maximum likelihood update.
incremental_ml_update	Incremental maximum likelihood update.
no_replacement	No replacement (static search)

Definition at line 23 of file it-ea.hh.

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

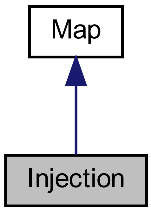
- lib/hnco/algorithms/evolutionary-algorithms/it-ea.hh
- lib/hnco/algorithms/evolutionary-algorithms/it-ea.cc

5.61 Injection Class Reference

Injection.

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

Inheritance diagram for Injection:



Public Member Functions

- Injection (const std::vector< int > &bit_positions, int output_size)
  Constructor.
- void **map** (const bit_vector_t &input, bit_vector_t &output) override
  Map
- int **get_input_size** () const override
  Get input size.
- int **get_output_size** () const override
  Get output size.
- bool **is_surjective** () const override
  Check for surjective map.

Private Attributes

- std::vector< int > **_bit_positions**

  Bit positions.
- int **_output_size**

  Output size.

5.61.1 Detailed Description

Injection.

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

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

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

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

Definition at line 493 of file map.hh.

5.61.2 Constructor & Destructor Documentation

5.61.2.1 Injection()

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

Constructor.

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

Parameters

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

Precondition

output_size >= bit_positions.size()

Definition at line 169 of file map.cc.

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

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

5.62 IntegerCategoricalRepresentation Class Reference

Integer categorical representation.

`#include <hnco/representations/categorical.hh>`

Public Types

- using **domain_type** = std::size_t
  Domain type.

Public Member Functions

- IntegerCategoricalRepresentation (int num_categories)
  Constructor.
- int **size** () const
  Size of the representation.
- domain_type **unpack** (const bit_vector_t &bv, int start)
  Unpack bit vector into a category.
- void **display** (std::ostream &stream) const
  Display.

Private Attributes

- int **_num_categories**
  Number of categories.
- int **_size**
  Size in bits.

5.62.1 Detailed Description

Integer categorical representation.

Definition at line 142 of file categorical.hh.

5.62.2 Constructor & Destructor Documentation

5.62.2.1 IntegerCategoricalRepresentation()

```
IntegerCategoricalRepresentation (
            int num_categories ) [inline]
```

Constructor.

Parameters

num_categories	Number of categories

Definition at line 160 of file categorical.hh.

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

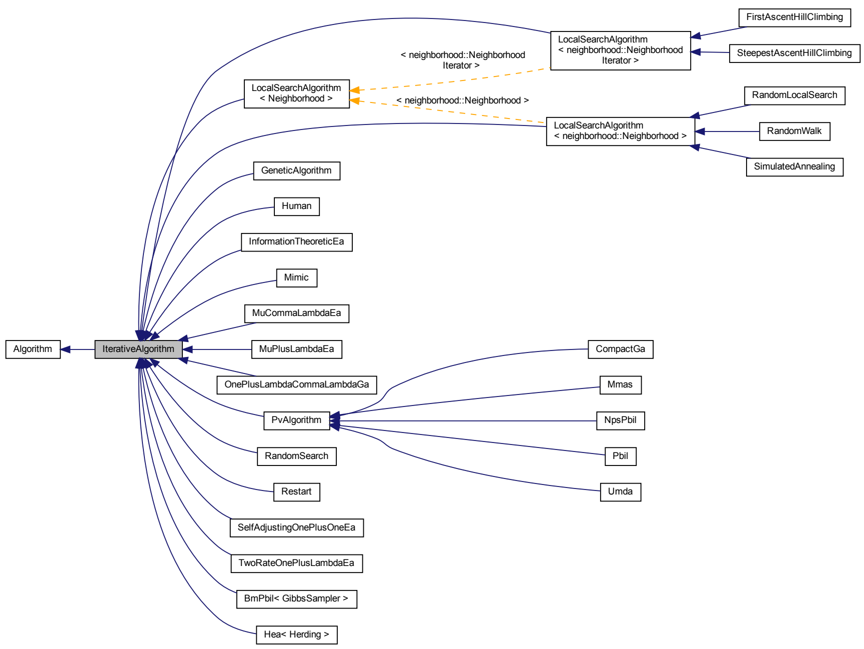
- lib/hnco/representations/categorical.hh

5.63 IterativeAlgorithm Class Reference

Iterative search.

`#include <hnco/algorithms/iterative-algorithm.hh>`

Inheritance diagram for IterativeAlgorithm:



Public Member Functions

- IterativeAlgorithm (int n)

  Constructor.

Optimization

- void maximize (const std::vector< function::Function ∗ > &functions) override

  Maximize.

Setters

- void set_num_iterations (int x)

  Set the number of iterations.

Protected Member Functions

Loop

- virtual void **init** ()
  Initialize.
- virtual void **iterate** ()=0
  Single iteration.
- virtual void **log** ()
  Log.
- virtual void loop () final
  Loop.

Protected Attributes

- int **_iteration**
  Current iteration.
- bool **_last_iteration** = false
  Last iteration.
- bool **_something_to_log** = false
  Something to log.

Parameters

- int **_num_iterations** = 0
  Number of iterations.

5.63.1 Detailed Description

Iterative search.

Definition at line 32 of file iterative-algorithm.hh.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 IterativeAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 83 of file iterative-algorithm.hh.

5.63.3 Member Function Documentation

5.63.3.1 loop()

```
void loop ( )  [final], [protected], [virtual]
```

Loop.

Calls init() then enter the main loop which, at each iteration, calls iterate() then log() only if _something_to_log is true.

Definition at line 28 of file iterative-algorithm.cc.

5.63.3.2 maximize()

```
void maximize (
            const std::vector< function::Function * > & functions )  [override], [virtual]
```

Maximize.

Calls set_functions() then loop.

Implements Algorithm.

Definition at line 53 of file iterative-algorithm.cc.

5.63.3.3 set_num_iterations()

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

Set the number of iterations.

Parameters

x	Number of iterations

Warning

x <= 0 means indefinite

Definition at line 109 of file iterative-algorithm.hh.

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

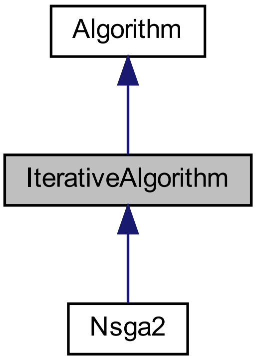
- lib/hnco/algorithms/iterative-algorithm.hh
- lib/hnco/algorithms/iterative-algorithm.cc

5.64 IterativeAlgorithm Class Reference

Iterative algorithm.

`#include <hnco/multiobjective/algorithms/iterative-algorithm.hh>`

Inheritance diagram for IterativeAlgorithm:



Public Member Functions

- IterativeAlgorithm (int n, int num_objectives)
  Constructor.

Optimization

- void minimize (const std::vector< Function * > &functions) override
  Minimize.

Setters

- void set_num_iterations (int n)
  Set the number of iterations.

Protected Member Functions

Loop

- virtual void **init** ()
  Initialize.
- virtual void **iterate** ()=0
  Single iteration.
- virtual void **log** ()
  Log.
- virtual void **finalize** ()
  Finalize.
- virtual void loop () final
  Loop.

Protected Attributes

- int **_iteration**
  Current iteration.
- bool **_last_iteration** = false
  Last iteration.
- bool **_something_to_log** = false
  Something to log.

Parameters

- int **_num_iterations** = 0
  Number of iterations.

Additional Inherited Members

5.64.1 Detailed Description

Iterative algorithm.

Definition at line 33 of file iterative-algorithm.hh.

5.64.2 Constructor & Destructor Documentation

5.64.2.1 IterativeAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors
num_objectives	Number of objectives

Definition at line 87 of file iterative-algorithm.hh.

5.64.3 Member Function Documentation

5.64.3.1 loop()

```
void loop ( )  [final], [protected], [virtual]
```

Loop.

Calls init() then enter the main loop which, at each iteration, calls iterate() then log() only if _something_to_log is true.

Definition at line 28 of file iterative-algorithm.cc.

5.64.3.2 minimize()

```
void minimize (
            const std::vector< Function * > & functions )  [override], [virtual]
```

Minimize.

Calls set_functions() then loop.

Implements Algorithm.

Definition at line 43 of file iterative-algorithm.cc.

5.64.3.3 set_num_iterations()

```
void set_num_iterations (
            int n )  [inline]
```

Set the number of iterations.

Parameters

n	Number of iterations

Warning

n <= 0 means indefinite

Definition at line 113 of file iterative-algorithm.hh.

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

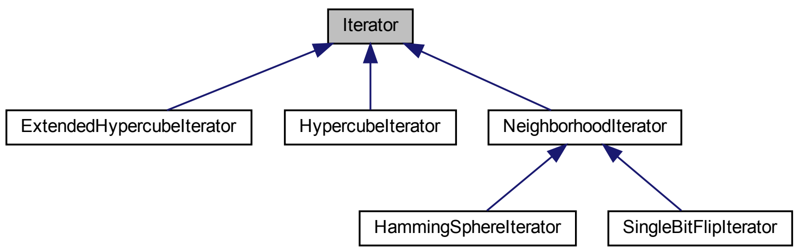
- lib/hnco/multiobjective/algorithms/iterative-algorithm.hh
- lib/hnco/multiobjective/algorithms/iterative-algorithm.cc

5.65 Iterator Class Reference

Iterator over bit vectors

`#include <hnco/iterator.hh>`

Inheritance diagram for Iterator:



Public Member Functions

- **Iterator** (int n)

  Constructor.
- virtual ∼**Iterator** ()

  Destructor.
- virtual void **init** ()

  Initialization.
- virtual bool **has_next** ()=0

  Has next bit vector.
- virtual const bit_vector_t & **next** ()=0

  Next bit vector.

Protected Attributes

- bit_vector_t **_current**

  Current bit vector.
- bool **_initial_state** = true

  Flag for initial state.

5.65.1 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:

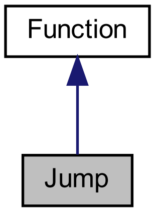
- lib/hnco/iterator.hh

5.66 Jump Class Reference

Jump.

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

Inheritance diagram for Jump:



Public Member Functions

- **Jump** (int bv_size, int gap)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Private Attributes

- int **_bv_size**
  Bit vector size.
- int **_gap**
  Gap.

5.66.1 Detailed Description

Jump.

Reference:

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

Definition at line 41 of file jump.hh.

5.66.2 Member Function Documentation

5.66.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 64 of file jump.hh.

5.66.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 60 of file jump.hh.

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

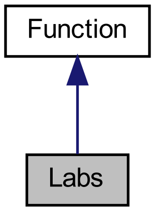
- lib/hnco/functions/collection/jump.hh
- lib/hnco/functions/collection/jump.cc

5.67 Labs Class Reference

Low autocorrelation binary sequences.

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

Inheritance diagram for Labs:



Public Member Functions

- **Labs** (int n)

    Constructor.
- void **set_merit_factor_flag** (bool b)

    Set merit factor flag.
- int **get_bv_size** () const override

    Get bit vector size.
- double **evaluate** (const bit_vector_t &) override

    Evaluate a bit vector.

Protected Member Functions

- double **compute_autocorrelation** (const bit_vector_t &)

    Compute autocorrelation.

Protected Attributes

- std::vector< int > **_sequence**

    Binary sequence written using 1 and -1.
- bool **_merit_factor_flag** = false

    Merit factor flag.

5.67.1 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

If _merit_factor_flag is true then the function returns n / (2 ∗ autocorrelation) else it returns -autocorrelation.

Definition at line 44 of file labs.hh.

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

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

5.68 LastEvaluation Class Reference

Last evaluation.

`#include <hnco/exception.hh>`

Inherits runtime_error.

5.68.1 Detailed Description

Last evaluation.

Definition at line 33 of file exception.hh.

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

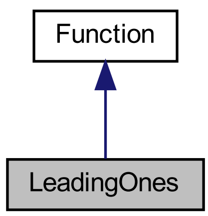
- lib/hnco/exception.hh

5.69 LeadingOnes Class Reference

Leading ones.

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

Inheritance diagram for LeadingOnes:



Public Member Functions

- **LeadingOnes** (int bv_size)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.

Private Attributes

- int **_bv_size**
  Bit vector size.

5.69.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 100 of file theory.hh.

5.69.2 Member Function Documentation

5.69.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 123 of file theory.hh.

5.69.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 119 of file theory.hh.

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

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

5.70 LinearCategoricalRepresentation Class Reference

Linear categorical representation.

`#include <hnco/representations/categorical.hh>`

Public Types

- using **domain_type** = std::size_t

  Domain type.

Public Member Functions

- LinearCategoricalRepresentation (int num_categories)

  Constructor.
- int **size** () const

  Size of the representation.
- domain_type **unpack** (const bit_vector_t &bv, int start)

  Unpack bit vector into a category.
- void **display** (std::ostream &stream) const

  Display.

Private Attributes

- int **_num_categories**

  Number of categories.
- int **_nrows**

  Number of rows.
- int **_ncols**

  Number of columns.
- bit_matrix_t **_A**

  Linear code as a bit matrix.
- bit_vector_t **_y**

  Output category.
- bit_vector_t **_x**

  Input bit vector.

5.70.1 Detailed Description

Linear categorical representation.

Definition at line 42 of file categorical.hh.

5.70.2 Constructor & Destructor Documentation

5.70.2.1 LinearCategoricalRepresentation()

```
LinearCategoricalRepresentation (
            int num_categories )  [inline]
```

Constructor.

Parameters

num_categories	Number of categories

Definition at line 72 of file categorical.hh.

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

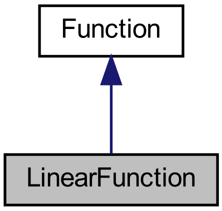
- lib/hnco/representations/categorical.hh

5.71 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



Public Member Functions

- **LinearFunction** ()

  Constructor.

Instance generators

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

  Instance generator.
- void random (int n)

  Random instance.

Load and save instance

- void load (std::string path)

  Load instance.
- void save (std::string path) const

  Save instance.

Evaluation

- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits) override

  Incrementally evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override

  Get bit vector size.
- double **get_maximum** () const override

  Get the global maximum.
- bool has_known_maximum () const override

  Check for a known maximum.
- bool provides_incremental_evaluation () const override

  Check whether the function provides incremental evaluation.
- void **display** (std::ostream &stream) const override

  Display.

Private Member Functions

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

  Serialize.

Private Attributes

- std::vector< double > **_weights**

  Weights.

5.71.1 Detailed Description

Linear function.

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

5.71.2 Member Function Documentation

5.71.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

Definition at line 70 of file linear-function.hh.

5.71.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 135 of file linear-function.hh.

5.71.2.3 load()

```
void load (
            std::string path ) [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

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

5.71.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 140 of file linear-function.hh.

5.71.2.5 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

n	Size of bit vectors

Definition at line 82 of file linear-function.hh.

5.71.2.6 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

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

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

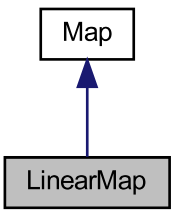
- lib/hnco/functions/collection/linear-function.hh
- lib/hnco/functions/collection/linear-function.cc

5.72 LinearMap Class Reference

Linear map.

`#include <hnco/maps/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) override

  Map
- int **get_input_size** () const override

  Get input size.
- int **get_output_size** () const override

  Get output size.
- bool is_surjective () const override

  Check for surjective map.

Load and save map

- void load (std::string path)

  Load map.
- void save (std::string path) const

  Save 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.

5.72.1 Detailed Description

Linear map.

A linear map f from F_2^m to F_2^n is defined by $f(x) = Ax$, where A is an n x m bit matrix.

Definition at line 248 of file map.hh.

5.72.2 Member Function Documentation

5.72.2.1 is_surjective()

```
bool is_surjective ( ) const  [override], [virtual]
```

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 105 of file map.cc.

5.72.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 311 of file map.hh.

5.72.2.3 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

std::runtime_error	

Definition at line 76 of file map.cc.

5.72.2.4 save()

```
void save (
            std::string path ) const  [inline]
```

Save map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 318 of file map.hh.

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

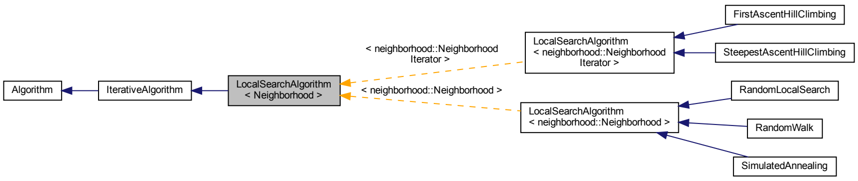
- lib/hnco/maps/map.hh
- lib/hnco/maps/map.cc

5.73 LocalSearchAlgorithm< Neighborhood > Class Template Reference

Local search algorithm.

```
#include <hnco/algorithms/local-search/local-search-algorithm.hh>
```

Inheritance diagram for LocalSearchAlgorithm< Neighborhood >:



Public Member Functions

- **LocalSearchAlgorithm** (int n, Neighborhood ∗neighborhood)
  Constructor.

Setters

- void **set_random_initialization** (bool b)
  Set random initialization.
- void **set_starting_point** (const bit_vector_t &x)
  Set the starting point.

Protected Member Functions

Loop

- void **init** () override
  Initialize.

Protected Attributes

- bit_vector_t **_starting_point**
  Starting point.
- Neighborhood ∗ **_neighborhood**
  Neighborhood.

Parameters

- bool **_random_initialization** = true
  Random initialization.

5.73.1 Detailed Description

template<class Neighborhood>
class hnco::algorithm::LocalSearchAlgorithm< Neighborhood >

Local search algorithm.

Definition at line 33 of file local-search-algorithm.hh.

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

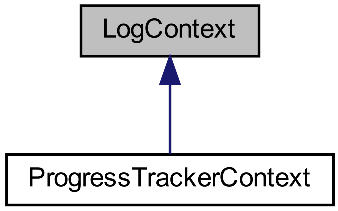
- lib/hnco/algorithms/local-search/local-search-algorithm.hh

5.74 LogContext Class Reference

Log context.

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

Inheritance diagram for LogContext:



Public Member Functions

- virtual std::string **to_string** ()=0

    Get context.

5.74.1 Detailed Description

Log context.

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

Definition at line 40 of file log-context.hh.

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

- lib/hnco/logging/log-context.hh

5.75 Logger Class Reference

Logger.

`#include <hnco/logging/logger.hh>`

Public Member Functions

- **Logger** ()
  Default constructor.
- Logger (LogContext ∗context)
  Constructor.
- std::ostringstream & **line** ()
  Get the line.
- virtual ∼Logger ()
  Destructor.

Static Public Member Functions

- static std::ostream & **stream** ()
  Get the stream.
- static void **set_stream** (std::ostream ∗stream)
  Set the stream.

Private Attributes

- std::ostringstream **_line**
  Line.

Static Private Attributes

- static std::ostream ∗ **_stream** = &std::cout
  Output stream.

5.75.1 Detailed Description

Logger.

Simple logger inspired by the Log class published in Dr. Dobb's:

`https://www.drdobbs.com/cpp/logging-in-c/201804215`

Definition at line 43 of file logger.hh.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 Logger()

```
Logger (
            LogContext * context )  [inline]
```

Constructor.

The constructor converts the context to a string which it writes at the beginning of the line.

Parameters

context	Log context

Definition at line 69 of file logger.hh.

5.75.2.2 ~Logger()

```
virtual ~Logger ( ) [inline], [virtual]
```

Destructor.

Send the line to the output stream and add an end of line.

Definition at line 81 of file logger.hh.

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

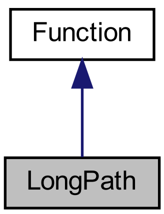
- lib/hnco/logging/logger.hh
- lib/hnco/logging/logger.cc

5.76 LongPath Class Reference

Long path.

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

Inheritance diagram for LongPath:



Public Member Functions

- **LongPath** (int bv_size, int prefix_length)
  Constructor.
- double **evaluate** (const bit_vector_t &)
  Evaluate a bit vector.

Information about the function

- int **get_bv_size** () const
  Get bit vector size.
- bool has_known_maximum () const
  Check for a known maximum.
- double get_maximum () const
  Get the global maximum.

Private Attributes

- int **_bv_size**
  Bit vector size.
- int **_prefix_length**
  Prefix length.

5.76.1 Detailed Description

Long path.

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

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

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

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

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

References:

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

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.76.2 Member Function Documentation

5.76.2.1 get_maximum()

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

Get the global maximum.

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

Exceptions

std::runtime_error	

Reimplemented from Function.

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

5.76.2.2 has_known_maximum()

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

Check for a known maximum.

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

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

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

Reimplemented from Function.

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

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

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

5.77 LowerTriangularWalshMoment2 Struct Reference

Lower triangular Walsh moment.

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

Public Member Functions

- LowerTriangularWalshMoment2 (int n)
  Constructor.
- void display (std::ostream &stream)
  Display Walsh moment.
- void **init** ()
  Initialize Walsh moment.
- void **add** (const bit_vector_t &bv)
  Add a bit vector to a Walsh moment.
- void **average** (int count)
  Average each Walsh moment.
- void update (const LowerTriangularWalshMoment2 &wm, double rate)
  Update a Walsh moment.
- void update (const LowerTriangularWalshMoment2 &wm1, const LowerTriangularWalshMoment2 &wm2, double rate)
  Update a Walsh moment.
- void scaled_difference (double lambda, const LowerTriangularWalshMoment2 &wm1, const LowerTriangularWalshMoment2 &wm2)
  Compute a scaled difference between two moments.
- void bound (double margin)
  Bound Walsh moment.
- double **norm_1** () const
  1-norm of the Walsh moment
- double **norm_2** () const
  2-norm of the Walsh moment
- double **norm_infinite** () const
  infinite-norm of the Walsh moment
- double **distance** (const LowerTriangularWalshMoment2 &wm) const
  distance between the Walsh moment and another Walsh moment

Public Attributes

- std::vector< double > **first_moment**
  First moment.
- std::vector< std::vector< double > > **second_moment**
  Second moment.

5.77.1 Detailed Description

Lower triangular Walsh moment.

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

5.77.2 Constructor & Destructor Documentation

5.77.2.1 LowerTriangularWalshMoment2()

```
LowerTriangularWalshMoment2 (
            int n )
```

Constructor.

Parameters

n	Size of bit vector

Definition at line 33 of file walsh-moment.cc.

5.77.3 Member Function Documentation

5.77.3.1 bound()

```
void bound (
            double margin )
```

Bound Walsh moment.

Ensure that the distance from each Walsh moment to the -1/1 bounds is greater or equal to the given margin.

Parameters

margin	Distance from the -1/1 bounds

Definition at line 162 of file walsh-moment.cc.

5.77.3.2 display()

```
void display (
            std::ostream & stream )
```

Display Walsh moment.

A LowerTriangularWalshMoment2 is displayed as a full symmetric matrix with diagonal entries equal to first moments and off-diagonal entries equal to second moments.

Definition at line 44 of file walsh-moment.cc.

5.77.3.3 scaled_difference()

```
void scaled_difference (
            double lambda,
            const LowerTriangularWalshMoment2 & wm1,
            const LowerTriangularWalshMoment2 & wm2 )
```

Compute a scaled difference between two moments.

This member function implements:

self = lambda $*$ wm1 - wm2

It is mostly useful in herding (Hea).

Parameters

lambda	Scale
wm1	First Walsh moment
wm2	Second Walsh moment

Definition at line 143 of file walsh-moment.cc.

5.77.3.4 update() [1/2]

```
void update (
            const LowerTriangularWalshMoment2 & wm,
            double rate )
```

Update a Walsh moment.

This member function implements:

self += rate ∗ (wm1 - self)

Parameters

wm	Target Walsh moment
rate	Learning rate

Postcondition

For all i, is_in_interval(first_moment[i], -1, 1)

For all j $<$ i, is_in_interval(second_moment[i][j], -1, 1)

Definition at line 105 of file walsh-moment.cc.

5.77.3.5 update() [2/2]

```
void update (
            const LowerTriangularWalshMoment2 & wm1,
            const LowerTriangularWalshMoment2 & wm2,
            double rate )
```

Update a Walsh moment.

This member function implements:

self += rate ∗ (wm1 - wm2)

The resulting entries are not necessarily those of a Walsh moment, that is

is_in_interval(first_moment[i], -1, 1) or

is_in_interval(second_moment[i][j], -1, 1)

might fail for some i, j.

Parameters

wm1	Target Walsh moment
wm2	Walsh moment to move away from
rate	Learning rate

Definition at line 123 of file walsh-moment.cc.

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

- lib/hnco/algorithms/walsh-moment/walsh-moment.hh
- lib/hnco/algorithms/walsh-moment/walsh-moment.cc

5.78 LowerTriangularWalshMoment2GibbsSampler Class Reference

Gibbs sampler with lower triangular Walsh moments.

```
#include <hnco/algorithms/walsh-moment/gibbs-sampler.hh>
```

Public Types

- using **Moment** = LowerTriangularWalshMoment2

  Walsh moment type.

Public Member Functions

- **LowerTriangularWalshMoment2GibbsSampler** (int n, const LowerTriangularWalshMoment2 &mp)

  Constructor.
- void **init** ()

  Initialize.
- void **update** (int i)

  Update state.
- void **update_sync** ()

  Update state synchronously.
- const bit_vector_t & **get_state** ()

  Get the state of the Gibbs sampler.

Private Attributes

- const LowerTriangularWalshMoment2 & **_model_parameters**

  Model parameters.
- bit_vector_t **_state**

  State of the Gibbs sampler.
- pv_t **_pv**

  Probability vector for synchronous Gibbs sampling.

5.78.1 Detailed Description

Gibbs sampler with lower triangular Walsh moments.

Definition at line 38 of file gibbs-sampler.hh.

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

- lib/hnco/algorithms/walsh-moment/gibbs-sampler.hh
- lib/hnco/algorithms/walsh-moment/gibbs-sampler.cc

5.79 LowerTriangularWalshMoment2Herding Class Reference

Herding with lower triangular Walsh moment.

```
#include <hnco/algorithms/walsh-moment/herding.hh>
```

Public Types

- using **Moment** = LowerTriangularWalshMoment2

    Walsh moment type.

Public Member Functions

- LowerTriangularWalshMoment2Herding (int n)

    Constructor.
- void **init** ()

    Initialization.
- void **sample** (const LowerTriangularWalshMoment2 &target, bit_vector_t &x)

    Sample a bit vector.
- double **error** (const LowerTriangularWalshMoment2 &target)

    Compute the error.

Getters

- const LowerTriangularWalshMoment2 & **get_delta** () const

    Get delta.

Setters

- void **set_randomize_bit_order** (bool x)

    Randomize bit order.

Protected Attributes

- LowerTriangularWalshMoment2 **_delta**

    Delta moment.
- LowerTriangularWalshMoment2 **_count**

    Counter moment.
- LowerTriangularWalshMoment2 **_error**

    Error moment.
- permutation_t **_permutation**

    Permutation.
- int **_time**

    Time.

Parameters

- bool **_randomize_bit_order** = false

    Randomize bit order.

5.79.1 Detailed Description

Herding with lower triangular Walsh moment.

Definition at line 37 of file herding.hh.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 LowerTriangularWalshMoment2Herding()

```
LowerTriangularWalshMoment2Herding (
              int n )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors

Definition at line 74 of file herding.hh.

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

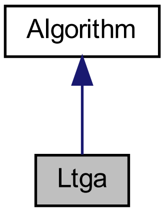
- lib/hnco/algorithms/walsh-moment/herding.hh
- lib/hnco/algorithms/walsh-moment/herding.cc

5.80 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.
- ∼**Ltga** ()
  Destructor.
- void **maximize** (const std::vector< function::Function ∗ > &functions)
  Maximize.
- void **finalize** ()
  Finalize.
- void **set_population_size** (int n)
  Set population size.

Private Attributes

- Implementation ∗ _implementation
  Pointer to implementation.
- int **_population_size** = 10
  Population size.

Additional Inherited Members

5.80.1 Detailed Description

Linkage Tree Genetic Algorithm.

Implementation of the Linkage Tree Genetic Algorithm.

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Reference:

"Hierarchical problem solving with the linkage tree genetic algorithm" by D. Thierens and P. A. N. Bosman

Definition at line 48 of file ltga.hh.

5.80.2 Member Data Documentation

5.80.2.1 _implementation

`Implementation* _implementation [private]`

Pointer to implementation.

The main motivation for this pattern is to avoid including declarations from fast_efficient_p3 into the global namespace.

A raw pointer is used instead of a unique_ptr because the latter will not compile with pybind11.

Definition at line 58 of file ltga.hh.

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

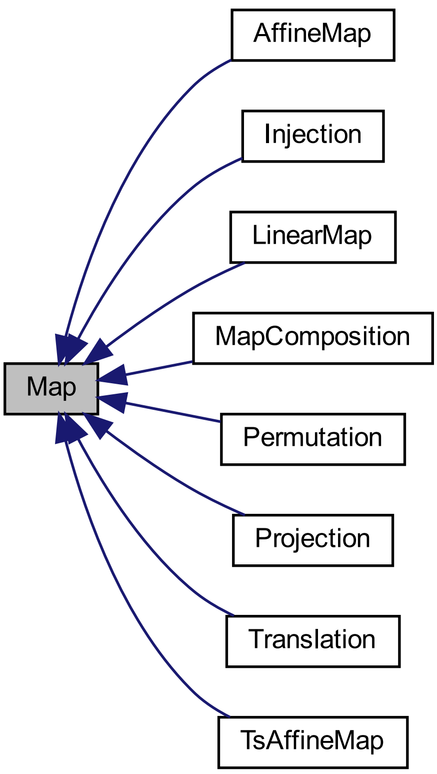
- lib/hnco/algorithms/fast-efficient-p3/ltga.hh
- lib/hnco/algorithms/fast-efficient-p3/ltga.cc

5.81 Map Class Reference

Map

`#include <hnco/maps/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 int **get_input_size** () const =0
  Get input size.
- virtual int **get_output_size** () const =0
  Get output size.
- virtual bool is_surjective () const
  Check for surjective map.
- virtual void **display** (std::ostream &stream) const
  Display.

5.81.1 Detailed Description

Map

Definition at line 46 of file map.hh.

5.81.2 Member Function Documentation

5.81.2.1 is_surjective()

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

Check for surjective map.

Returns

false

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

Definition at line 66 of file map.hh.

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

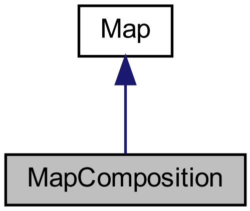
- lib/hnco/maps/map.hh

5.82 MapComposition Class Reference

Map composition.

```
#include <hnco/maps/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) override
  Map
- int **get_input_size** () const override
  Get input size.
- int **get_output_size** () const override
  Get output size.
- bool is_surjective () const override
  Check for surjective map.

Private Attributes

- Map ∗ **_outer**
  Outer map.
- Map ∗ **_inner**
  Inner map.
- bit_vector_t **_bv**
  Temporary bit vector.

5.82.1 Detailed Description

Map composition.

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

Definition at line 424 of file map.hh.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 MapComposition()

```
MapComposition (
            Map * outer,
            Map * inner )  [inline]
```

Constructor.

Parameters

outer	outer map
inner	inner map

Precondition

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

Definition at line 448 of file map.hh.

5.82.3 Member Function Documentation

5.82.3.1 is_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 472 of file map.hh.

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

- lib/hnco/maps/map.hh

5.83 MapgenOptions Class Reference

Command line options for mapgen.

```
#include <mapgen-options.hh>
```

Public Member Functions

- **MapgenOptions** ()
  Default constructor.
- **MapgenOptions** (int argc, char ∗argv[ ], bool ignore_bad_options=false)
  Constructor.
- int **get_input_size** () const
  Get the value of input_size.
- bool **with_input_size** () const
  With parameter input_size.
- int **get_map** () const
  Get the value of map.
- bool **with_map** () const
  With parameter map.
- int **get_output_size** () const
  Get the value of output_size.
- bool **with_output_size** () const
  With parameter output_size.
- std::string **get_path** () const
  Get the value of path.
- bool **with_path** () const
  With parameter path.
- int **get_seed** () const
  Get the value of seed.
- bool **with_seed** () const
  With parameter seed.
- int **get_ts_length** () const
  Get the value of ts_length.
- bool **with_ts_length** () const
  With parameter ts_length.
- int **get_ts_sampling_mode** () const
  Get the value of ts_sampling_mode.
- bool **with_ts_sampling_mode** () const
  With parameter ts_sampling_mode.
- bool **with_surjective** () const
  With the flag surjective.

Private Member Functions

- void **print_help** (std::ostream &stream) const
  Print help message.
- void **print_version** (std::ostream &stream) const
  Print version.

Private Attributes

- std::string **_exec_name**
  Name of the executable.
- std::string **_version** = "0.24"
  Name Version.
- int **_input_size** = 100
  Input bit vector size.
- int **_map** = 1
  Type of map.
- int **_output_size** = 100
  Output bit vector size.
- std::string **_path** = "map.txt"
  Path (relative or absolute) of a map file.
- int **_seed**
  Seed for the random number generator.
- int **_ts_length** = 10
  Transvection sequence length.
- int **_ts_sampling_mode** = 0
  Transvection sequence sampling mode.
- bool **_surjective** = false
  Ensure that the sampled linear or affine map is surjective.

Friends

- std::ostream & **operator**<< (std::ostream &, const MapgenOptions &)
  Print a header containing the parameter values.

5.83.1 Detailed Description

Command line options for mapgen.

Definition at line 11 of file mapgen-options.hh.

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

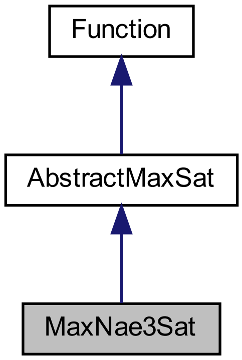
- app/mapgen-options.hh
- app/mapgen-options.cc

5.84 MaxNae3Sat Class Reference

Max not-all-equal 3SAT.

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

Inheritance diagram for MaxNae3Sat:



Public Member Functions

- **MaxNae3Sat** ()

  Default constructor.
- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.
- void load (std::string path)

  Load instance.

Additional Inherited Members

5.84.1 Detailed Description

Max not-all-equal 3SAT.

Reference:

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

Definition at line 162 of file max-sat.hh.

5.84.2 Member Function Documentation

5.84.2.1 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 177 of file max-sat.hh.

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

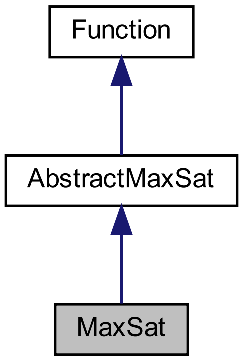
- lib/hnco/functions/collection/max-sat.hh
- lib/hnco/functions/collection/max-sat.cc

5.85 MaxSat Class Reference

MAX-SAT.

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

Inheritance diagram for MaxSat:



Public Member Functions

- **MaxSat** ()
  Default constructor.
- void random (int n, int k, int c)
  Random instance.
- void random (const bit_vector_t &solution, int k, int c)
  Random instance with satisfiable expression.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Additional Inherited Members

5.85.1 Detailed Description

MAX-SAT.

Reference:

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

Definition at line 119 of file max-sat.hh.

5.85.2 Member Function Documentation

5.85.2.1 random() [1/2]

```
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
c	Number of clauses

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

5.85.2.2 random() [2/2]

```
void random (
            int n,
            int k,
            int c )
```

Random instance.

Parameters

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

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

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

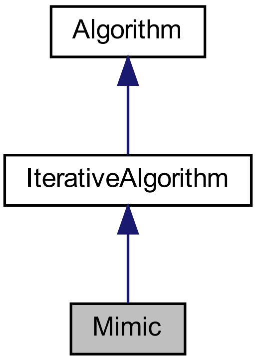
- lib/hnco/functions/collection/max-sat.hh
- lib/hnco/functions/collection/max-sat.cc

5.86 Mimic Class Reference

Mutual information maximizing input clustering.

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

Inheritance diagram for Mimic:



Public Member Functions

- **Mimic** (int n, int population_size)
  Constructor.

Setters

- void **set_selection_size** (int selection_size)
  Set the selection size.

Protected Member Functions

- void **sample** (bit_vector_t &bv)
  Sample a bit vector.
- void **compute_conditional_entropy** (int index)
  Compute conditional entropy.
- void **update_model** ()
  Update model.

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- Population **_population**
  Population.
- permutation_t **_permutation**
  Permutation.
- std::array< pv_t, 2 > **_parameters**
  Model parameters.
- pv_t **_mean**
  Mean of selected bit vectors.
- std::vector< double > **_entropies**
  Conditional entropies.
- std::array< std::array< int, 2 >, 2 > **_table**
  Contingency table.
- double **_lower_bound**
  Lower bound of probability.
- double **_upper_bound**
  Upper bound of probability.

Parameters

- int **_selection_size**
  Selection size.

5.86.1 Detailed Description

Mutual information maximizing input clustering.

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

Reference:

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

Definition at line 52 of file mimic.hh.

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

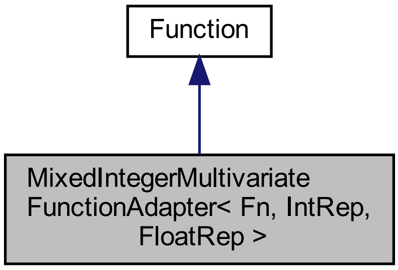
- lib/hnco/algorithms/mimic.hh
- lib/hnco/algorithms/mimic.cc

5.87 MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep > Class Template Reference

Mixed-integer multivariate function adapter.

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

Inheritance diagram for MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep >:



Public Types

- using **function_type** = Fn
  Function type
- using **int_rep_type** = IntRep
  Integer type.
- using **float_rep_type** = FloatRep
  Float type.

Public Member Functions

- MixedIntegerMultivariateFunctionAdapter (Fn ∗fn, std::vector< IntRep > int_reps, std::vector< FloatRep > float_reps, std::vector< std::pair< bool, int > > lut)
  Constructor.

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.

Evaluation

- double **evaluate** (const bit_vector_t &bv) override
  Evaluate.

Display

- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override
  Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)

  Unpack a bit vector into values.

Private Attributes

- Fn * **_function**

  Multivariate function.
- std::vector< IntRep > **_int_reps**

  Integer representations.
- std::vector< FloatRep > **_float_reps**

  Float representations.
- std::vector< typename Fn::domain_type > **_variables**

  Variables.
- std::vector< std::pair< bool, int > > **_lut**

  Lookup table.

5.87.1 Detailed Description

template<typename Fn, typename IntRep, typename FloatRep>
class hnco::function::MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep >

Mixed-integer multivariate function adapter.

The purpose of this class is to build a regular hnco function from an arbitrary multivariate function. This is achieved using a composition:

- Representations (Rep): hypercube -> domain
- Multivariate function (Fn): product of domains -> double

Definition at line 172 of file multivariate-function-adapter.hh.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 MixedIntegerMultivariateFunctionAdapter()

```
MixedIntegerMultivariateFunctionAdapter (
            Fn * fn,
            std::vector< IntRep > int_reps,
            std::vector< FloatRep > float_reps,
            std::vector< std::pair< bool, int > > lut )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
int_reps	Integer representations
float_reps	Float representations
lut	Lookup table

For each variable, the lookup table tells whether it is an integer or a float, and gives its index in the corresponding representation table, _int_reps or _float_reps.

Definition at line 238 of file multivariate-function-adapter.hh.

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

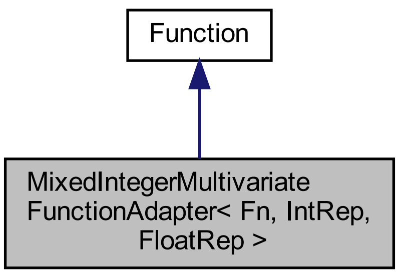
- lib/hnco/functions/multivariate-function-adapter.hh

5.88 MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep > Class Template Reference

Mixed-integer multivariate function adapter.

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

Inheritance diagram for MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep >:



Public Types

- using **function_type** = Fn

    Function type.
- using **int_rep_type** = IntRep

    Integer type.
- using **float_rep_type** = FloatRep

    Float type.

Public Member Functions

- MixedIntegerMultivariateFunctionAdapter (Fn ∗fn, std::vector< IntRep > int_reps, std::vector< FloatRep > float_reps, std::vector< std::pair< bool, int > > lut)

    Constructor.

Information about the function

- int **get_bv_size** () const override

    Get bit vector size.
- int **get_output_size** () const override

    Get output size (number of objectives)

Evaluation

- void **evaluate** (const bit_vector_t &bv, value_t &value) override

    Evaluate.

Display

- void **display** (std::ostream &stream) const override

    Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override

    Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)

    Unpack a bit vector into values.

Private Attributes

- Fn ∗ **_function**

    Multivariate function.
- std::vector< IntRep > **_int_reps**

    Integer representations.
- std::vector< FloatRep > **_float_reps**

    Float representations.
- std::vector< typename Fn::domain_type > **_variables**

    Variables.
- std::vector< std::pair< bool, int > > **_lut**

    Lookup table.

5.88.1 Detailed Description

template<typename Fn, typename IntRep, typename FloatRep>
class hnco::multiobjective::function::MixedIntegerMultivariateFunctionAdapter< Fn, IntRep, FloatRep >

Mixed-integer multivariate function adapter.

The purpose of this class is to build a regular hnco function from an arbitrary multivariate function. This is achieved using a composition:

- Representations (Rep): hypercube -> domain
- Multivariate function (Fn): product of domains -> product of codomains (double)

Definition at line 189 of file multivariate-function-adapter.hh.

5.88.2 Constructor & Destructor Documentation

5.88.2.1 MixedIntegerMultivariateFunctionAdapter()

```
MixedIntegerMultivariateFunctionAdapter (
            Fn * fn,
            std::vector< IntRep > int_reps,
            std::vector< FloatRep > float_reps,
            std::vector< std::pair< bool, int > > lut )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
int_reps	Integer representations
float_reps	Float representations
lut	Lookup table

For each variable, the lookup table tells whether it is an integer or a float, and gives its index in the corresponding representation table, _int_reps or _float_reps.

Definition at line 255 of file multivariate-function-adapter.hh.

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

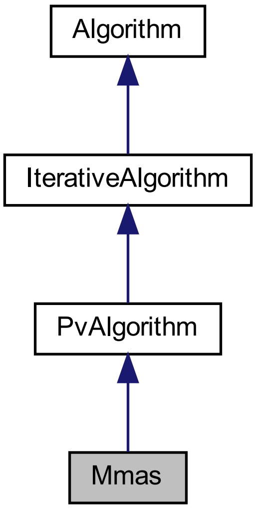
- lib/hnco/multiobjective/functions/multivariate-function-adapter.hh

5.89 Mmas Class Reference

Max-min ant system.

```
#include <hnco/algorithms/probability-vector/mmas.hh>
```

Inheritance diagram for Mmas:



Public Member Functions

- **Mmas** (int n)
  Constructor.

Setters

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

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- bit_vector_t **_x**
  Candidate solution.

Parameters

- std::function< bool(double, double)> **_compare** = std::greater_equal<double>()
  Binary operator for comparing evaluations.
- double **_learning_rate** = 1e-3
  Learning rate.

5.89.1 Detailed Description

Max-min ant system.

Reference:

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

Definition at line 42 of file mmas.hh.

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

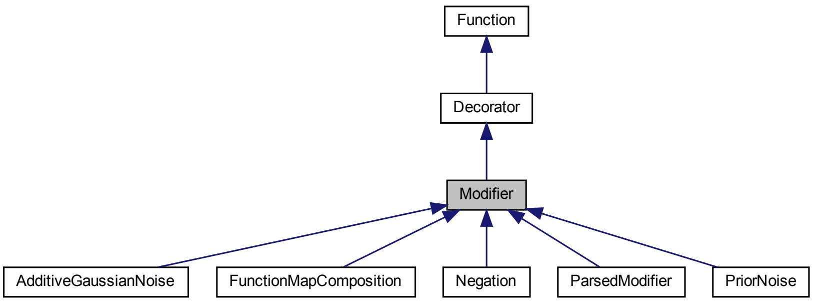
- lib/hnco/algorithms/probability-vector/mmas.hh
- lib/hnco/algorithms/probability-vector/mmas.cc

5.90 Modifier Class Reference

Function modifier.

`#include <hnco/functions/modifiers/modifier.hh>`

Inheritance diagram for Modifier:



Public Member Functions

- **Modifier** (Function ∗function)

  Constructor.

Additional Inherited Members

5.90.1 Detailed Description

Function modifier.

Definition at line 39 of file modifier.hh.

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

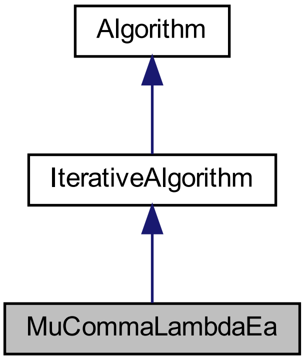
- lib/hnco/functions/modifiers/modifier.hh

5.91 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



Public Member Functions

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

    Constructor.

Setters

- void **set_mutation_rate** (double p)

    Set the mutation rate.
- void **set_allow_no_mutation** (bool b)

    Set the flag _allow_no_mutation.

Protected Member Functions

Loop

- void **init** () override

    Initialize.
- void **iterate** () override

    Single iteration.

Protected Attributes

- Population **_parents**

  Parents.
- Population **_offsprings**

  Offsprings.
- CommaSelection **_comma_selection**

  Comma selection.
- neighborhood::StandardBitMutation **_mutation**

  Mutation operator.
- std::uniform_int_distribution< int > **_select_parent**

  Select parent.

Parameters

- double **_mutation_rate**

  Mutation rate.
- bool **_allow_no_mutation** = false

  Allow no mutation.

5.91.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 43 of file mu-comma-lambda-ea.hh.

5.91.2 Constructor & Destructor Documentation

5.91.2.1 MuCommaLambdaEa()

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

Constructor.

Parameters

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

Definition at line 94 of file mu-comma-lambda-ea.hh.

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

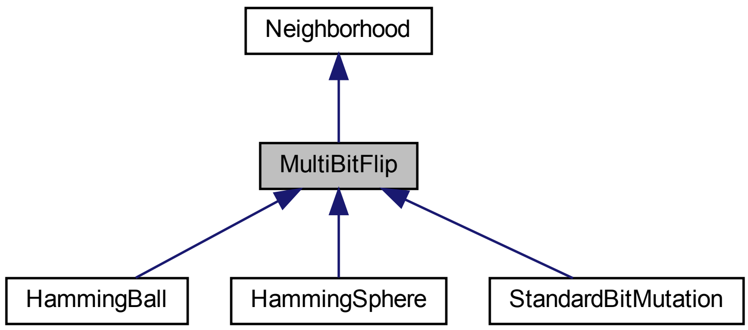
- lib/hnco/algorithms/evolutionary-algorithms/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/evolutionary-algorithms/mu-comma-lambda-ea.cc

5.92 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 rejection_sampling (int k)

  Sample a given number of bits using rejection sampling.

Additional Inherited Members

5.92.1 Detailed Description

Multi bit flip.

Definition at line 185 of file neighborhood.hh.

5.92.2 Constructor & Destructor Documentation

5.92.2.1 MultiBitFlip()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 208 of file neighborhood.hh.

5.92.3 Member Function Documentation

5.92.3.1 bernoulli_trials()

```
void bernoulli_trials (
            int k )  [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.

5.92.3.2 rejection_sampling()

```
void rejection_sampling (
            int k )  [protected]
```

Sample a given number of bits using rejection 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:

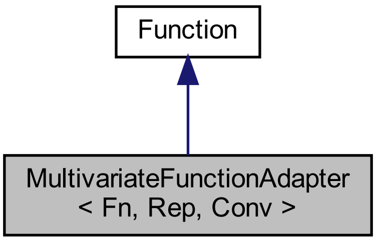
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.93 MultivariateFunctionAdapter< Fn, Rep, Conv > Class Template Reference

Multivariate function adapter.

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

Inheritance diagram for MultivariateFunctionAdapter< Fn, Rep, Conv >:



Public Types

- using **function_type** = Fn

    Function type
- using **representation_type** = Rep

    Representation type.
- using **converter_type** = Conv

    Converter type.

Public Member Functions

- MultivariateFunctionAdapter (Fn ∗fn, std::vector< Rep > reps)

    Constructor.

Information about the function

- int **get_bv_size** () const override

    Get bit vector size.

Evaluation

- double **evaluate** (const bit_vector_t &bv) override
  Evaluate.

Display

- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override
  Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)
  Unpack a bit vector into values.

Private Attributes

- Fn * **_function**
  Multivariate function.
- std::vector< Rep > **_representations**
  Representations.
- std::vector< typename Fn::domain_type > **_variables**
  Variables.
- Conv **_converter**
  Converter from codomain to double.

5.93.1 Detailed Description

template<class Fn, class Rep, class Conv>
class hnco::function::MultivariateFunctionAdapter< Fn, Rep, Conv >

Multivariate function adapter.

The purpose of this class is to build a regular hnco function from an arbitrary multivariate function. This is achieved using a composition:

- Representations (Rep): hypercube -> domain
- Multivariate function (Fn): product of domains -> codomain
- Converter (Conv): codomain -> double

Definition at line 47 of file multivariate-function-adapter.hh.

5.93.2 Constructor & Destructor Documentation

5.93.2.1 MultivariateFunctionAdapter()

```
MultivariateFunctionAdapter (
            Fn * fn,
            std::vector< Rep > reps )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
reps	Representations

Definition at line 96 of file multivariate-function-adapter.hh.

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

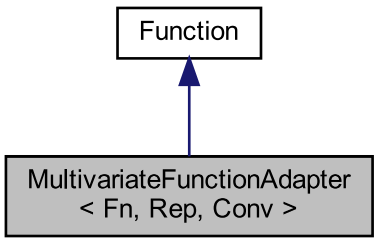
- lib/hnco/functions/multivariate-function-adapter.hh

5.94 MultivariateFunctionAdapter< Fn, Rep, Conv > Class Template Reference

Multivariate function adapter.

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

Inheritance diagram for MultivariateFunctionAdapter< Fn, Rep, Conv >:



Public Types

- using **function_type** = Fn

    Function type.
- using **representation_type** = Rep

    Representation type.
- using **converter_type** = Conv

    Converter type.

Public Member Functions

- MultivariateFunctionAdapter (Fn ∗fn, std::vector< Rep > reps)

    Constructor.

Information about the function

- int **get_bv_size** () const override

    Get bit vector size.
- int **get_output_size** () const override

    Get output size (number of objectives)

Evaluation

- void **evaluate** (const bit_vector_t &bv, value_t &value) override

    Evaluate.

Display

- void **display** (std::ostream &stream) const override

    Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override

    Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)

    Unpack a bit vector into variables.

Private Attributes

- Fn ∗ **_function**

    Multivariate function.
- std::vector< Rep > **_representations**

    Representations.
- std::vector< typename Fn::domain_type > **_variables**

    Variables.
- std::vector< typename Fn::codomain_type > **_codomain_value**

    Codomain value.
- Conv **_converter**

    Converter from codomain to double.

5.94.1 Detailed Description

template<class Fn, class Rep, class Conv>
class hnco::multiobjective::function::MultivariateFunctionAdapter< Fn, Rep, Conv >

Multivariate function adapter.

The purpose of this class is to build a regular hnco function from an arbitrary multivariate function. This is achieved using a composition:

- Representations (Rep): hypercube -> domain
- Multivariate function (Fn): product of domains -> product of codomains
- Converter (Conv): codomain -> double

Definition at line 49 of file multivariate-function-adapter.hh.

5.94.2 Constructor & Destructor Documentation

5.94.2.1 MultivariateFunctionAdapter()

```
MultivariateFunctionAdapter (
            Fn * fn,
            std::vector< Rep > reps )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
reps	Representations

Definition at line 102 of file multivariate-function-adapter.hh.

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

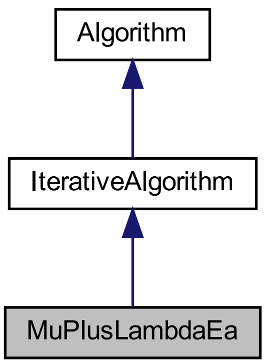
- lib/hnco/multiobjective/functions/multivariate-function-adapter.hh

5.95 MuPlusLambdaEa Class Reference

(mu+lambda) EA.

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

Inheritance diagram for MuPlusLambdaEa:



Public Member Functions

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

  Constructor.

Setters

- void **set_mutation_rate** (double p)

  Set the mutation rate.
- void **set_allow_no_mutation** (bool b)

  Set the flag _allow_no_mutation.

Protected Member Functions

Loop

- void **init** () override

  Initialize.
- void **iterate** () override

  Single iteration.

Protected Attributes

- Population **_parents**

  Parents.
- Population **_offsprings**

  Offsprings.
- PlusSelection **_plus_selection**

  Plus selection.
- neighborhood::StandardBitMutation **_mutation**

  Mutation operator.
- std::uniform_int_distribution< int > **_select_parent**

  Select parent.

Parameters

- double **_mutation_rate**

  Mutation rate.
- bool **_allow_no_mutation** = false

  Allow no mutation.

5.95.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 43 of file mu-plus-lambda-ea.hh.

5.95.2 Constructor & Destructor Documentation

5.95.2.1 MuPlusLambdaEa()

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

Constructor.

Parameters

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

Definition at line 94 of file mu-plus-lambda-ea.hh.

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

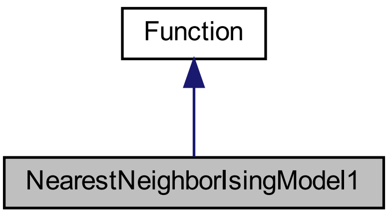
- lib/hnco/algorithms/evolutionary-algorithms/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/evolutionary-algorithms/mu-plus-lambda-ea.cc

5.96 NearestNeighborIsingModel1 Class Reference

Nearest neighbor Ising model in one dimension.

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

Inheritance diagram for NearestNeighborIsingModel1:



Public Member Functions

- **NearestNeighborIsingModel1** ()
  Constructor.
- void **set_periodic_boundary_conditions** (bool x)
  Set periodic boundary conditions.

Instance generators

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

Load and save instance

- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

Evaluation

- double evaluate (const bit_vector_t &) override
  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits) override
  Incrementally evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.
- bool provides_incremental_evaluation () const override
  Check whether the function provides incremental evaluation.
- void **display** (std::ostream &stream) const override
  Display.

Private Member Functions

- template<class Archive >
  void **save** (Archive &ar, const unsigned int version) const
  Save.
- template<class Archive >
  void **load** (Archive &ar, const unsigned int version)
  Load.
- void **resize** (int n)
  Resize data structures.

Private Attributes

- std::vector< double > **_coupling**

  Coupling with nearest neighbor to the right.
- std::vector< double > **_field**

  External field.
- bit_vector_t **_flipped_bits**

  Flipped bits.
- bool **_periodic_boundary_conditions** = false

  Periodic boundary conditions.

5.96.1 Detailed Description

Nearest neighbor Ising model in one dimension.

Its expression is of the form

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

or equivalently

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

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

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

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

It should be noted that such an Ising model can be represented by a Walsh expansion of degree 2, that is WalshExpansion2.

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

Definition at line 63 of file nearest-neighbor-ising-model-1.hh.

5.96.2 Member Function Documentation

5.96.2.1 evaluate()

```
double evaluate (
            const bit_vector_t & s )  [override], [virtual]
```

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.96.2.2 generate()

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

Instance generator.

Parameters

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

Definition at line 124 of file nearest-neighbor-ising-model-1.hh.

5.96.2.3 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 158 of file nearest-neighbor-ising-model-1.hh.

5.96.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 199 of file nearest-neighbor-ising-model-1.hh.

5.96.2.5 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

n	Size of bit vector

Definition at line 140 of file nearest-neighbor-ising-model-1.hh.

5.96.2.6 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 165 of file nearest-neighbor-ising-model-1.hh.

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

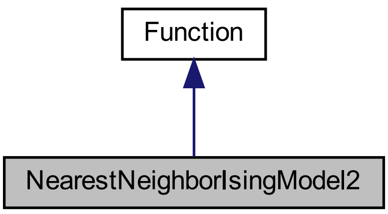
- lib/hnco/functions/collection/ising/nearest-neighbor-ising-model-1.hh
- lib/hnco/functions/collection/ising/nearest-neighbor-ising-model-1.cc

5.97 NearestNeighborIsingModel2 Class Reference

Nearest neighbor Ising model in two dimensions.

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

Inheritance diagram for NearestNeighborIsingModel2:



Public Member Functions

- **NearestNeighborIsingModel2** ()
  Constructor.
- void **set_periodic_boundary_conditions** (bool x)
  Set periodic boundary conditions.

Instance generators

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

Load and save instance

- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

Evaluation

- double evaluate (const bit_vector_t &) override
  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits) override
  Incrementally evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.
- bool provides_incremental_evaluation () const override
  Check whether the function provides incremental evaluation.
- void **display** (std::ostream &stream) const override
  Display.

Private Member Functions

- template<class Archive >
  void **save** (Archive &ar, const unsigned int version) const
  Save.
- template<class Archive >
  void **load** (Archive &ar, const unsigned int version)
  Load.
- void **resize** (int num_rows, int num_columns)
  Resize data structures.

Private Attributes

- std::vector< std::vector< double > > **_coupling_right**

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

  Coupling with nearest neighbor below.
- std::vector< std::vector< double > > **_field**

  External field.
- bit_vector_t **_flipped_bits**

  Flipped bits.
- bool **_periodic_boundary_conditions** = false

  Periodic boundary conditions.

5.97.1 Detailed Description

Nearest neighbor Ising model in two dimensions.

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

The expression of the function is of the form

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

or equivalently

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

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

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

It should be noted that such an Ising model can be represented by a Walsh expansion of degree 2, that is WalshExpansion2.

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

Definition at line 65 of file nearest-neighbor-ising-model-2.hh.

5.97.2 Member Function Documentation

5.97.2.1 evaluate()

```
double evaluate (
            const bit_vector_t & s )  [override], [virtual]
```

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.97.2.2 generate()

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

Instance generator.

Parameters

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

Definition at line 132 of file nearest-neighbor-ising-model-2.hh.

5.97.2.3 load()

```
void load (
            std::string path ) [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 170 of file nearest-neighbor-ising-model-2.hh.

5.97.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 216 of file nearest-neighbor-ising-model-2.hh.

5.97.2.5 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

num_rows	Number of rows
num_columns	Number of columns

Definition at line 152 of file nearest-neighbor-ising-model-2.hh.

5.97.2.6 save()

```
void save (
            std::string path ) const [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 177 of file nearest-neighbor-ising-model-2.hh.

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

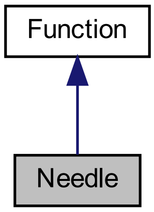
- lib/hnco/functions/collection/ising/nearest-neighbor-ising-model-2.hh
- lib/hnco/functions/collection/ising/nearest-neighbor-ising-model-2.cc

5.98 Needle Class Reference

Needle in a haystack.

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

Inheritance diagram for Needle:



Public Member Functions

- **Needle** (int bv_size)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.

Private Attributes

- int **_bv_size**
  Bit vector size.

5.98.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 135 of file theory.hh.

5.98.2 Member Function Documentation

5.98.2.1 get_maximum()

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

Get the global maximum.

Returns

1

Reimplemented from Function.

Definition at line 158 of file theory.hh.

5.98.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 154 of file theory.hh.

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

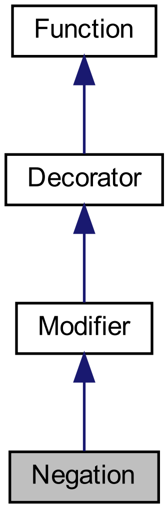
- lib/hnco/functions/collection/theory.hh
- lib/hnco/functions/collection/theory.cc

5.99 Negation Class Reference

Negation.

```
#include <hnco/functions/modifiers/modifier.hh>
```

Inheritance diagram for Negation:



Public Member Functions

- **Negation** (Function ∗function)

  Constructor.

Information about the function

- int **get_bv_size** () const override

  Get bit vector size.
- bool provides_incremental_evaluation () const override

  Check whether the function provides incremental evaluation.

Evaluation

- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped_bits) override

  Incrementally evaluate a bit vector.

Additional Inherited Members

5.99.1 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 60 of file modifier.hh.

5.99.2 Member Function Documentation

5.99.2.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 79 of file modifier.hh.

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

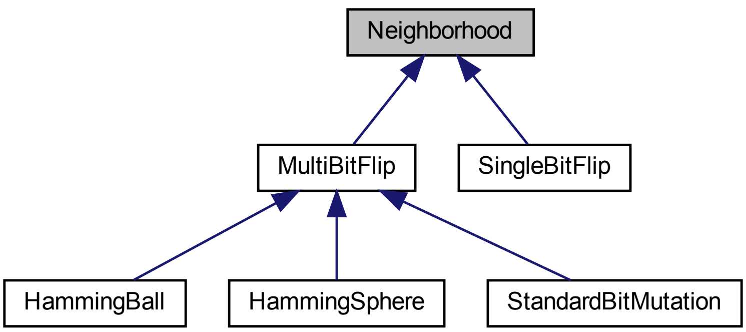
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.100 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** () const

    Get the origin.
- virtual const bit_vector_t & **get_candidate** () const

    Get the candidate bit vector.
- virtual const sparse_bit_vector_t & **get_flipped_bits** () const

    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)

    Map.

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
- std::uniform_int_distribution< int > **_index_dist**

    Index distribution.
- sparse_bit_vector_t **_flipped_bits**

    Flipped bits.

5.100.1 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.

5.100.2 Constructor & Destructor Documentation

5.100.2.1 Neighborhood()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 86 of file neighborhood.hh.

5.100.3 Member Function Documentation

5.100.3.1 map()

```
virtual void map (
            const bit_vector_t & input,
            bit_vector_t & output )  [inline], [virtual]
```

Map.

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 151 of file neighborhood.hh.

5.100.3.2 mutate()

```
virtual void mutate (
            bit_vector_t & bv ) [inline], [virtual]
```

Mutate.

In-place mutation of the bit vector.

Parameters

bv	Bit vector to mutate

Definition at line 137 of file neighborhood.hh.

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

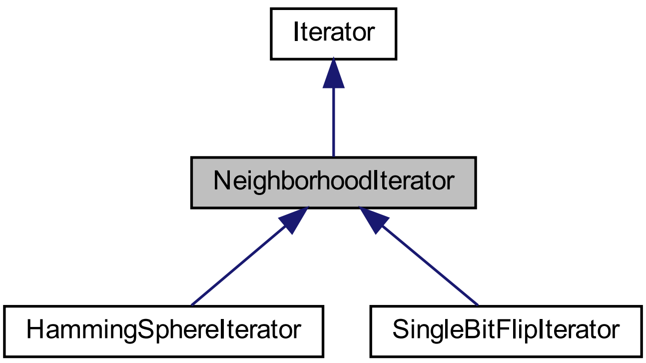
- lib/hnco/neighborhoods/neighborhood.hh

5.101 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

5.101.1 Detailed Description

Neighborhood iterator.

A neighborhood iterator allows to iterate over bit vectors in the neighborhood of a given origin. The origin itself should not belong to the neighborhood.

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

5.101.2 Constructor & Destructor Documentation

5.101.2.1 NeighborhoodIterator()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 47 of file neighborhood-iterator.hh.

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

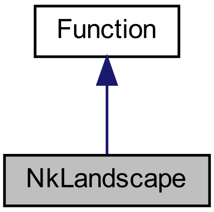
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.102 NkLandscape Class Reference

NK landscape.

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

Inheritance diagram for NkLandscape:



Public Member Functions

- **NkLandscape** ()

  Default constructor.
- int **get_bv_size** () const override

  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override

  Evaluate a bit vector.
- void **display** (std::ostream &stream) const override

  Display.

Instance generators

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

  Instance generator.
- void random (int n, int k)

  Random instance.

Load and save instance

- void load (std::string path)

  Load instance.
- void save (std::string path) const

  Save instance.

Private Member Functions

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

  Serialize.
- void random_structure (int n, int k)

  Random structue.

Private Attributes

- std::vector< std::vector< int > > **_neighbors**

  Bit neighbors.
- std::vector< std::vector< double > > **_partial_functions**

  Partial functions.

5.102.1 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 45 of file nk-landscape.hh.

5.102.2 Member Function Documentation

5.102.2.1 generate()

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

Instance generator.

Parameters

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

Definition at line 89 of file nk-landscape.hh.

5.102.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 126 of file nk-landscape.hh.

5.102.2.3 random()

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

Random instance.

Partial function values are sampled from the normal distribution.

Parameters

n	Size of bit vector
k	Number of neighbors per bit

Definition at line 107 of file nk-landscape.hh.

5.102.2.4 random_structure()

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

Random structue.

Parameters

n	Size of bit vector
k	Number of neighbors per bit

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

5.102.2.5 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 133 of file nk-landscape.hh.

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

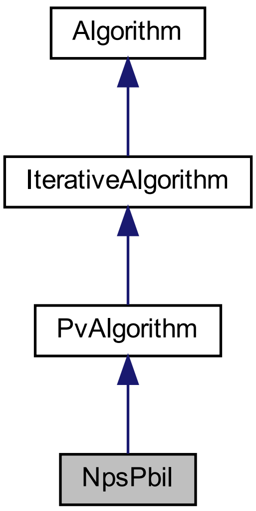
- lib/hnco/functions/collection/nk-landscape.hh
- lib/hnco/functions/collection/nk-landscape.cc

5.103 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

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

Inheritance diagram for NpsPbil:



Public Member Functions

- **NpsPbil** (int n, int population_size)
  Constructor.

Setters

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

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- Population **_population**
  Population.
- pv_t **_mean_best**
  Mean of best individuals.
- pv_t **_mean_worst**
  Mean of worst individuals.

Parameters

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

5.103.1 Detailed Description

Population-based incremental learning with negative and positive selection.

Reference:

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

Definition at line 42 of file nps-pbil.hh.

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

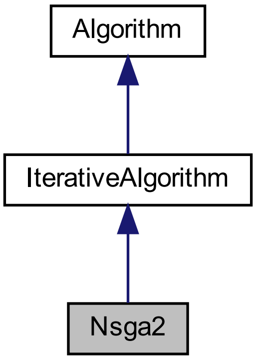
- lib/hnco/algorithms/probability-vector/nps-pbil.hh
- lib/hnco/algorithms/probability-vector/nps-pbil.cc

5.104 Nsga2 Class Reference

NSGA-II.

`#include <hnco/multiobjective/algorithms/nsga2.hh>`

Inheritance diagram for Nsga2:



Public Member Functions

- Nsga2 (int n, int num_objectives, int population_size)
  Constructor.
- const Population & **get_solutions** () override
  Get solutions.

Setters

- void **set_tournament_size** (int n)
  Set the tournament size.
- void **set_mutation_rate** (double p)
  Set the mutation rate.
- void **set_allow_no_mutation** (bool b)
  Set the flag _allow_no_mutation.
- void **set_crossover_probability** (double p)
  Set the crossover probability.

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.
- void **finalize** () override
  Finalize.
- void **log** () override
  Log.

Protected Attributes

- Population **_parents**
  Parent population.
- Population **_offsprings**
  Offspring population.
- Population **_full_population**
  Full population.
- Population **_solutions**
  Solutions.
- neighborhood::StandardBitMutation **_mutation**
  Mutation operator.
- std::bernoulli_distribution **_do_crossover**
  Do crossover.
- hnco::algorithm::UniformCrossover **_crossover**
  Uniform crossover.
- Nsga2ParetoFrontComputation **_pareto_front_computation**
  Pareto front computation.
- std::vector< int > **_pareto_fronts**
  Pareto fronts.
- std::vector< double > **_crowding_distances**
  Crowding distances.
- hnco::permutation_t **_permutation**
  Permutation relative to Pareto front.
- std::vector< FrontDistancePair > **_front_distance_pairs**
  Front distance pairs.
- TournamentSelection< FrontDistancePair, std::less< FrontDistancePair > > **_selection_by_front_distance_pair**
  Selection by front distance pairs.

Parameters

- int **_tournament_size** = 2
  Tournament size.
- double **_mutation_rate**
  Mutation rate.
- bool **_allow_no_mutation** = false
  Allow no mutation.
- double **_crossover_probability** = 0.8
  Crossover probability.

Additional Inherited Members

5.104.1 Detailed Description

NSGA-II.

NSGA-II is a (mu+mu) evolutionary algorithm for multiobjective optimization.

Deb, Agrawal, Pratap, and Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic %Algorithm for Multi-objective Optimization: NSGA-II", Parallel Problem Solving from Nature PPSN VI, 2000, Springer Berlin Heidelberg.

`https://link.springer.com/chapter/10.1007/3-540-45356-3_83`

Definition at line 84 of file nsga2.hh.

5.104.2 Constructor & Destructor Documentation

5.104.2.1 Nsga2()

```
Nsga2 (
            int n,
            int num_objectives,
            int population_size )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors
num_objectives	Number of objectives
population_size	Population size

Definition at line 174 of file nsga2.hh.

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

- lib/hnco/multiobjective/algorithms/nsga2.hh
- lib/hnco/multiobjective/algorithms/nsga2.cc

5.105 Nsga2ParetoFrontComputation Class Reference

Pareto front computation from the NSGA-II paper.

`#include <hnco/multiobjective/algorithms/pareto-front-computation.hh>`

Public Member Functions

- **Nsga2ParetoFrontComputation** (Population &population)
  Constructor.
- void compute (std::vector< int > &pareto_fronts)
  Compute Pareto fronts.

Private Member Functions

- bool is_non_dominated (int i)
  Check that a value is non dominated.

Private Attributes

- const Population & **_population**

  Population
- std::vector< int > **_pool**

  Pool of values to consider for inclusion in the Pareto front.
- std::vector< int > **_next_pool**

  Next pool of values.
- std::unordered_set< int > **_non_dominated**

  Non dominated values.
- std::vector< int > _dominated

  Dominated values.

5.105.1 Detailed Description

Pareto front computation from the NSGA-II paper.

Definition at line 40 of file pareto-front-computation.hh.

5.105.2 Member Function Documentation

5.105.2.1 compute()

```
void compute (
            std::vector< int > & pareto_fronts )  [inline]
```

Compute Pareto fronts.

Parameters

pareto_fronts	Pareto fronts (output parameter)

Definition at line 89 of file pareto-front-computation.hh.

5.105.2.2 is_non_dominated()

```
bool is_non_dominated (
            int i )  [inline], [private]
```

Check that a value is non dominated.

Check that no value in the non dominated set dominates the considered value.

Parameters

i	Index of the value

Definition at line 67 of file pareto-front-computation.hh.

5.105.3 Member Data Documentation

5.105.3.1 _dominated

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

Dominated values.

To be removed from the non dominated ones.

Definition at line 58 of file pareto-front-computation.hh.

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

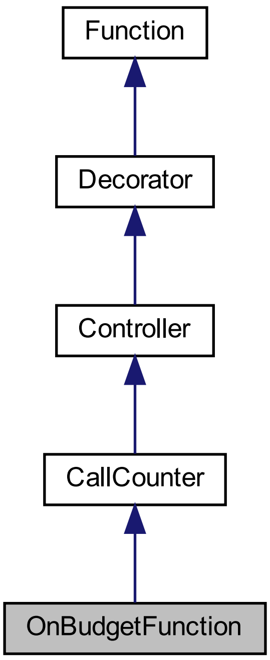
- lib/hnco/multiobjective/algorithms/pareto-front-computation.hh

5.106 OnBudgetFunction Class Reference

Function with a limited number of evaluations.

```
#include <hnco/functions/controllers/controller.hh>
```

Inheritance diagram for OnBudgetFunction:



Public Member Functions

- **OnBudgetFunction** (Function ∗function, int budget)

  Constructor.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &bv, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &bv, double value)

  Update after a safe evaluation.

Private Attributes

- int **_budget**

  Budget.

Additional Inherited Members

5.106.1 Detailed Description

Function with a limited number of evaluations.

Definition at line 195 of file controller.hh.

5.106.2 Member Function Documentation

5.106.2.1 evaluate()

```
double evaluate (
            const bit_vector_t & bv )  [virtual]
```

Evaluate a bit vector.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 96 of file controller.cc.

5.106.2.2 evaluate_incrementally()

```
double evaluate_incrementally (
            const bit_vector_t & bv,
            double value,
            const hnco::sparse_bit_vector_t & flipped_bits )  [virtual]
```

Incrementally evaluate a bit vector.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 105 of file controller.cc.

5.106.2.3 update()

```
void update (
            const bit_vector_t & bv,
            double value )  [virtual]
```

Update after a safe evaluation.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 114 of file controller.cc.

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

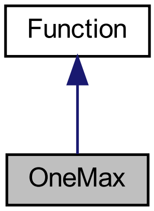
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.107 OneMax Class Reference

OneMax.

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

Inheritance diagram for OneMax:



Public Member Functions

- **OneMax** (int bv_size)

    Constructor.

Information about the function

- int **get_bv_size** () const override

    Get bit vector size.
- double get_maximum () const override

    Get the global maximum.
- bool has_known_maximum () const override

    Check for a known maximum.
- bool provides_incremental_evaluation () const override

    Check whether the function provides incremental evaluation.
- void **display** (std::ostream &stream) const override

    Display.

Evaluation

- double **evaluate** (const bit_vector_t &) override

    Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits) override

    Incrementally evaluate a bit vector.

Private Attributes

- int **_bv_size**

    Bit vector size.

5.107.1 Detailed Description

OneMax.

References:

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

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 41 of file theory.hh.

5.107.2 Member Function Documentation

5.107.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 61 of file theory.hh.

5.107.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 65 of file theory.hh.

5.107.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 70 of file theory.hh.

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

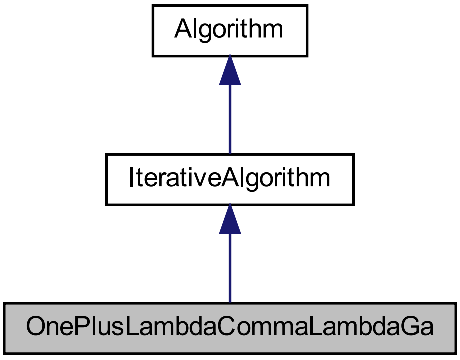
- lib/hnco/functions/collection/theory.hh
- lib/hnco/functions/collection/theory.cc

5.108 OnePlusLambdaCommaLambdaGa Class Reference

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

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

Inheritance diagram for OnePlusLambdaCommaLambdaGa:



Public Member Functions

- OnePlusLambdaCommaLambdaGa (int n, int lambda)
  Constructor.

Setters

- void **set_mutation_rate** (double p)
  Set the mutation rate.
- void **set_crossover_bias** (double x)
  Set the crossover bias.

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected 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_rate**
  Mutation rate.
- double **_crossover_bias**
  Crossover bias.

5.108.1 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.

5.108.2 Constructor & Destructor Documentation

5.108.2.1 OnePlusLambdaCommaLambdaGa()

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

Constructor.

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

Parameters

n	Size of bit vectors
lambda	Offspring population size

Definition at line 102 of file one-plus-lambda-comma-lambda-ga.hh.

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

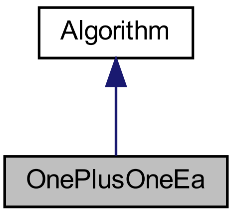
- lib/hnco/algorithms/evolutionary-algorithms/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/evolutionary-algorithms/one-plus-lambda-comma-lambda-ga.cc

5.109 OnePlusOneEa Class Reference

(1+1) EA.

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

Inheritance diagram for OnePlusOneEa:



Public Member Functions

- OnePlusOneEa (int n)
  Constructor.
- void **maximize** (const std::vector< function::Function * > &functions) override
  Maximize.
- void **finalize** () override
  Finalize.

Setters

- void set_num_iterations (int x)
  Set the number of iterations.
- void **set_mutation_rate** (double p)
  Set the mutation rate.
- void **set_allow_no_mutation** (bool b)
  Set the flag _allow_no_mutation.
- void **set_incremental_evaluation** (bool x)
  Set incremental evaluation.

Private Attributes

- neighborhood::StandardBitMutation **_neighborhood**

  Neighborhood.
- RandomLocalSearch **_rls**

  Random local search.

Parameters

- int **_num_iterations** = 0

  Number of iterations.
- double **_mutation_rate**

  Mutation rate.
- bool **_allow_no_mutation** = false

  Allow no mutation.
- bool **_incremental_evaluation** = false

  Incremental evaluation.

Additional Inherited Members

5.109.1 Detailed Description

(1+1) EA.

(1+1) EA is implemented as a RandomLocalSearch with a StandardBitMutation neighborhood and infinite patience. Thus the class OnePlusOneEa is derived from Algorithm instead of IterativeAlgorithm.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.109.2 Constructor & Destructor Documentation

5.109.2.1 OnePlusOneEa()

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

Constructor.

Parameters

n	Size of bit vectors

_mutation_rate is initialized to 1 / n.

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

5.109.3 Member Function Documentation

5.109.3.1 set_num_iterations()

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

Set the number of iterations.

Parameters

x	Number of iterations

x <= 0 means indefinite

Definition at line 111 of file one-plus-one-ea.hh.

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

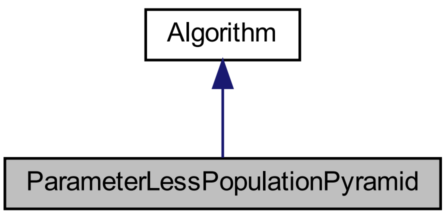
- lib/hnco/algorithms/evolutionary-algorithms/one-plus-one-ea.hh

5.110 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.
- ∼**ParameterLessPopulationPyramid** ()
  Destructor.
- void **maximize** (const std::vector< function::Function ∗ > &functions)
  Maximize.
- void **finalize** ()
  Finalize.

Private Attributes

- Implementation ∗ _implementation
  Pointer to implementation.

Additional Inherited Members

5.110.1 Detailed Description

Parameter-less Population Pyramid.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Reference:

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

Definition at line 51 of file p3.hh.

5.110.2 Member Data Documentation

5.110.2.1 _implementation

```
Implementation* _implementation [private]
```

Pointer to implementation.

The main motivation for this pattern is to avoid including declarations from fast_efficient_p3 into the global namespace.

A raw pointer is used instead of a unique_ptr because the latter will not compile with pybind11.

Definition at line 61 of file p3.hh.

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

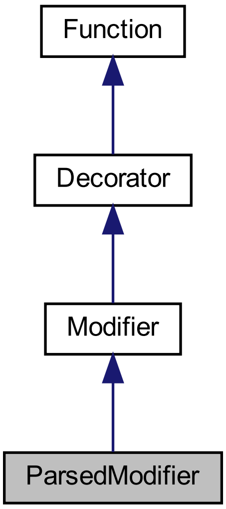
- lib/hnco/algorithms/fast-efficient-p3/p3.hh
- lib/hnco/algorithms/fast-efficient-p3/p3.cc

5.111 ParsedModifier Class Reference

Parsed modifier.

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

Inheritance diagram for ParsedModifier:



Public Member Functions

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

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.

Evaluation

- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Private Attributes

- FunctionParser **_fparser**
  Function parser.
- double **_values** [1]
  Array of values.

Additional Inherited Members

5.111.1 Detailed Description

Parsed modifier.

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

Definition at line 40 of file parsed-modifier.hh.

5.111.2 Constructor & Destructor Documentation

5.111.2.1 ParsedModifier()

```
ParsedModifier (
            Function * function,
            std::string expression )
```

Constructor.

Parameters

function	Decorated function
expression	Expression to parse

Definition at line 31 of file parsed-modifier.cc.

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

- lib/hnco/functions/modifiers/parsed-modifier.hh
- lib/hnco/functions/modifiers/parsed-modifier.cc

5.112 ParsedMultivariateFunction< Parser > Class Template Reference

Parsed multivariate function.

`#include <hnco/functions/collection/parsed-multivariate-function.hh>`

Public Types

- using **domain_type** = typename Parser::value_type
    Domain type.
- using **codomain_type** = typename Parser::value_type
    Codomain type.

Public Member Functions

- ParsedMultivariateFunction (std::string expression)
  Constructor.
- bool **add_constant** (std::string name, domain_type value)
  Add a constant to the parser.
- void **parse** ()
  Parse the expression.
- void **display** (std::ostream &stream) const
  Display the problem.
- codomain_type **evaluate** (const std::vector< domain_type > &x)
  Evaluate.
- void **describe** (const std::vector< domain_type > &x, std::ostream &stream)
  Describe a solution.
- int **get_num_variables** ()
  Get the number of variables.
- const std::vector< std::string > & **get_variable_names** ()
  Get variable names.

Private Attributes

- Parser **_fparser**
  Function parser
- std::vector< std::string > **_variable_names**
  Variable names.
- std::string **_expression**
  Expression.

5.112.1 Detailed Description

template< class Parser >
class hnco::function::ParsedMultivariateFunction< Parser >

Parsed multivariate function.

Uses the C++ library "Function Parser" (fparser):

`http://warp.povusers.org/FunctionParser/fparser.html`

Warning

The function string syntax depends on the chosen parser.

Definition at line 49 of file parsed-multivariate-function.hh.

5.112.2 Constructor & Destructor Documentation

5.112.2.1 ParsedMultivariateFunction()

```
ParsedMultivariateFunction (
        std::string expression ) [inline]
```

Constructor.

Parameters

expression	Expression to parse

Definition at line 72 of file parsed-multivariate-function.hh.

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

- lib/hnco/functions/collection/parsed-multivariate-function.hh

5.113 ParsedMultivariateFunction< Parser > Class Template Reference

Parsed multivariate function.

```
#include <hnco/multiobjective/functions/collection/parsed-multivariate-function.↩
hh>
```

Public Types

- using **domain_type** = typename Parser::value_type

    Domain type.
- using **codomain_type** = domain_type

    Codomain type.

Public Member Functions

- ParsedMultivariateFunction (std::string expression)

    Constructor.
- void **add_constant** (std::string name, domain_type value)

    Add a constant to the parsers.
- void **parse** ()

    Parse the expression.
- int **get_num_variables** () const

    Get the number of variables.
- int **get_output_size** () const

    Get output size (number of objectives)
- void **evaluate** (const std::vector< domain_type > &xs, std::vector< codomain_type > &values)

    Evaluate.
- void **display** (std::ostream &stream) const

    Display the problem.
- void **describe** (const std::vector< domain_type > &xs, std::ostream &stream)

    Describe a solution.
- const std::vector< std::string > & **get_variable_names** ()

    Get variable names.

Private Attributes

- std::vector< std::string > **_expressions**

  Expressions.
- std::vector< Parser > **_parsers**

  Function parsers
- std::vector< std::vector< std::string > > _names

  Names.
- std::vector< std::vector< domain_type > > _variables

  Variables.
- std::vector< std::vector< int > > _indices

  Indices.
- std::vector< std::string > _ordered_names

  Ordered variable names.

5.113.1 Detailed Description

template< class Parser >
class hnco::multiobjective::function::ParsedMultivariateFunction< Parser >

Parsed multivariate function.

Uses the C++ library "Function Parser" (fparser):

`http://warp.povusers.org/FunctionParser/fparser.html`

Warning

The function string syntax depends on the chosen parser.

Definition at line 54 of file parsed-multivariate-function.hh.

5.113.2 Constructor & Destructor Documentation

5.113.2.1 ParsedMultivariateFunction()

```
ParsedMultivariateFunction (
            std::string expression )  [inline]
```

Constructor.

An expression is a list of sub expressions separated by double colons (::). Each sub expression defines a multivariate function.

Parameters

expression	Expression to parse

Definition at line 114 of file parsed-multivariate-function.hh.

5.113.3 Member Data Documentation

5.113.3.1 _indices

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

Indices.

Indexed by parser then variable. Then, _indices[i][j] is the index in the vector to evaluate of the jth variable of the ith parser.

Definition at line 95 of file parsed-multivariate-function.hh.

5.113.3.2 _names

```
std::vector<std::vector<std::string> > _names [private]
```

Names.

Indexed by parser then variable. Then, _names[i][j] is the name of the jth variable of the ith parser.

Definition at line 78 of file parsed-multivariate-function.hh.

5.113.3.3 _ordered_names

```
std::vector<std::string> _ordered_names [private]
```

Ordered variable names.

As expected by evaluate().

Definition at line 102 of file parsed-multivariate-function.hh.

5.113.3.4 _variables

```
std::vector<std::vector<domain_type> > _variables [private]
```

Variables.

Indexed by parser then variable. Then, _variables[i][j] is the value of the jth variable of the ith parser.

Definition at line 86 of file parsed-multivariate-function.hh.

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

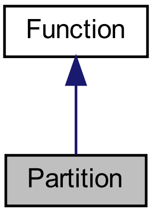
- lib/hnco/multiobjective/functions/collection/parsed-multivariate-function.hh

5.114 Partition Class Reference

Partition.

`#include <hnco/functions/collection/partition.hh>`

Inheritance diagram for Partition:



Public Member Functions

- **Partition** ()
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Instance generators

- template<class Generator >
  void generate (int n, Generator generator)
  Instance generator.
- void random (int n, int upper_bound)
  Random instance.

Load and save instance

- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

Display

- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &x, std::ostream &stream) override
  Describe a bit vector.

Private Member Functions

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

  Serialize.

Private Attributes

- std::vector< int > **_numbers**

  Multiset of positive integers.

5.114.1 Detailed Description

Partition.

Partition a finite multiset of positive integers into two subsets such that the sum of numbers in the first subset is the closest to the sum of numbers in the second subset.

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

Definition at line 52 of file partition.hh.

5.114.2 Member Function Documentation

5.114.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Number generator

Definition at line 84 of file partition.hh.

5.114.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 120 of file partition.hh.

5.114.2.3 random()

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

Random instance.

The numbers are sampled from the uniform distribution on [1..upper_bound].

Parameters

n	Size of bit vector
upper_bound	Upper bound of positive integers

Definition at line 100 of file partition.hh.

5.114.2.4 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 127 of file partition.hh.

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

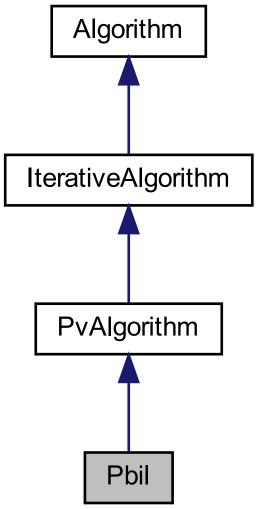
- lib/hnco/functions/collection/partition.hh
- lib/hnco/functions/collection/partition.cc

5.115 Pbil Class Reference

Population-based incremental learning.

```
#include <hnco/algorithms/probability-vector/pbil.hh>
```

Inheritance diagram for Pbil:



Public Member Functions

- **Pbil** (int n, int population_size)

  Constructor.

Setters

- void **set_selection_size** (int x)

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

  Set the learning rate.

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- Population **_population**
  Population.
- pv_t **_mean**
  Mean of selected bit vectors.

Parameters

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

5.115.1 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 42 of file pbil.hh.

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

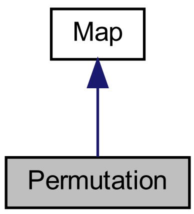
- lib/hnco/algorithms/probability-vector/pbil.hh
- lib/hnco/algorithms/probability-vector/pbil.cc

5.116 Permutation Class Reference

Permutation.

```
#include <hnco/maps/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) override

    Map
- int **get_input_size** () const override

    Get input size.
- int **get_output_size** () const override

    Get output size.
- bool is_surjective () const override

    Check for surjective map.

Load and save map

- void load (std::string path)

    Load map.
- void save (std::string path) const

    Save map.

Private Member Functions

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

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

    Load.

Private Attributes

- permutation_t **_permutation**

    Permutation.

5.116.1 Detailed Description

Permutation.

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

Definition at line 167 of file map.hh.

5.116.2 Member Function Documentation

5.116.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 218 of file map.hh.

5.116.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 229 of file map.hh.

5.116.2.3 save()

```
void save (
            std::string path ) const  [inline]
```

Save map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 236 of file map.hh.

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

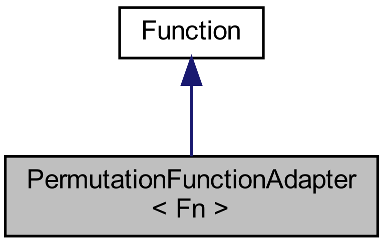
- lib/hnco/maps/map.hh
- lib/hnco/maps/map.cc

5.117 PermutationFunctionAdapter< Fn > Class Template Reference

Permutation function adapter.

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

Inheritance diagram for PermutationFunctionAdapter< Fn >:



Public Member Functions

- PermutationFunctionAdapter (Fn ∗fn, representation::PermutationRepresentation rep)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &bv) override
  Evaluate.
- void **display** (std::ostream &stream) const override
  Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override
  Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)
  Unpack a bit vector into a permutation.

Private Attributes

- Fn ∗ **_function**

  Permutation function.
- representation::PermutationRepresentation **_representation**

  Permutation representation.
- permutation_t **_permutation**

  Permutation.

5.117.1 Detailed Description

template<class Fn>
class hnco::function::PermutationFunctionAdapter< Fn >

Permutation function adapter.

The purpose of this class is to build a regular hnco function from an arbitrary function over permutations. This is achieved using a permutation representation.

Definition at line 42 of file permutation-function-adapter.hh.

5.117.2 Constructor & Destructor Documentation

5.117.2.1 PermutationFunctionAdapter()

```
PermutationFunctionAdapter (
            Fn * fn,
            representation::PermutationRepresentation rep )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
rep	Permutation representation

Definition at line 66 of file permutation-function-adapter.hh.

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

- lib/hnco/functions/permutation-function-adapter.hh

5.118 PermutationRepresentation Class Reference

Permutation representation.

```
#include <hnco/representations/permutation.hh>
```

Public Member Functions

- PermutationRepresentation (int num_elements, int num_additional_bits)

  Constructor.
- int **get_num_elements** () const

  Get number of elements.
- int **size** () const

  Size of the representation.
- void **unpack** (const bit_vector_t &bv, int start, hnco::permutation_t &permutation)

  Unpack bit vector into a permutation.
- void **display** (std::ostream &stream) const

  Display.

Private Attributes

- std::vector< int > **_values**

  Values to be sorted.
- int **_element_size**

  Element size in bits.
- int **_size**

  Size in bits.

5.118.1 Detailed Description

Permutation representation.

Definition at line 39 of file permutation.hh.

5.118.2 Constructor & Destructor Documentation

5.118.2.1 PermutationRepresentation()

```
PermutationRepresentation (
            int num_elements,
            int num_additional_bits )   [inline]
```

Constructor.

Each element is represented by an integer encoded using std::ceil(std::log(num_elements) / std::log(2)) + num_↩
additional_bits.

Parameters

num_elements	Number of elements
num_additional_bits	Number of additional bits per element

Definition at line 62 of file permutation.hh.

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

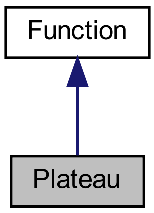
- lib/hnco/representations/permutation.hh

5.119 Plateau Class Reference

Plateau.

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

Inheritance diagram for Plateau:



Public Member Functions

- **Plateau** (int bv_size)

    Constructor.
- int **get_bv_size** () const override

    Get bit vector size.
- double **evaluate** (const bit_vector_t &) override

    Evaluate a bit vector.
- bool has_known_maximum () const override

    Check for a known maximum.
- double get_maximum () const override

    Get the global maximum.

Private Attributes

- int **_bv_size**

    Bit vector size.

5.119.1 Detailed Description

Plateau.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 242 of file theory.hh.

5.119.2 Member Function Documentation

5.119.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size + 2

Reimplemented from Function.

Definition at line 265 of file theory.hh.

5.119.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 261 of file theory.hh.

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

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

5.120 PlusSelection Class Reference

Plus selection.

```
#include <hnco/algorithms/evolutionary-algorithms/selection.hh>
```

Public Member Functions

- PlusSelection (Population &parents, Population &offsprings)
  Constructor.
- void **select** ()
  Apply selection.

Private Attributes

- Population & **_parents**
  Parent population.
- Population & **_offsprings**
  Offspring population.
- Population **_pool**
  Union of parent and offspring population.

5.120.1 Detailed Description

Plus selection.

Used as selection for replacement in evolutionary algorithms.

Definition at line 78 of file selection.hh.

5.120.2 Constructor & Destructor Documentation

5.120.2.1 PlusSelection()

```
PlusSelection (
            Population & parents,
            Population & offsprings )  [inline]
```

Constructor.

Parameters

parents	Parent population
offsprings	Offspring population

Definition at line 96 of file selection.hh.

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

- lib/hnco/algorithms/evolutionary-algorithms/selection.hh

5.121 Population Struct Reference

Population

```
#include <hnco/algorithms/population.hh>
```

Public Types

- using **Function** = hnco::function::Function
  Function type

Public Member Functions

- Population (int population_size, int n)
  Constructor.
- int **get_size** () const
  Get population size.
- int **get_bv_size** () const
  Get bit vector size.
- void **random** ()
  Sample a random population.

Get sorted bit vectors

- bit_vector_t & get_best_bv ()
  Get best bit vector.
- bit_vector_t & get_best_bv (int i)
  Get best bit vector.
- bit_vector_t & get_worst_bv (int i)
  Get worst bit vector.

Get sorted values

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

Evaluation and sorting

- void **evaluate** (Function ∗function)
  Evaluate the population.
- void **evaluate_in_parallel** (const std::vector< Function ∗ > &functions)
  Evaluate the population in parallel.
- void sort ()
  Sort the population.
- void partial_sort (int selection_size)
  Partially sort the population.
- std::pair< int, int > get_equivalent_bvs (int index)
  Get equivalent bit vectors.

Public Attributes

- std::vector< bit_vector_t > **bvs**
  Bit vectors.
- std::vector< double > **values**
  Values.
- hnco::permutation_t **permutation**
  Permutation.

5.121.1 Detailed Description

Population

Definition at line 41 of file population.hh.

5.121.2 Constructor & Destructor Documentation

5.121.2.1 Population()

```
Population (
            int population_size,
            int n ) [inline]
```

Constructor.

Parameters

population_size	Population size
n	Bit vector size

Definition at line 60 of file population.hh.

5.121.3 Member Function Documentation

5.121.3.1 get_best_bv() [1/2]

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

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 90 of file population.hh.

5.121.3.2 get_best_bv() [2/2]

```
bit_vector_t & get_best_bv (
            int i ) [inline]
```

Get best bit vector.

Parameters

i	Index in the sorted population

Precondition

The population must be sorted.

Definition at line 97 of file population.hh.

5.121.3.3 get_best_value() [1/2]

```
double get_best_value ( ) const [inline]
```

Get best value.

Precondition

The population must be sorted.

Definition at line 124 of file population.hh.

5.121.3.4 get_best_value() [2/2]

```
double get_best_value (
            int i ) const [inline]
```

Get best value.

Parameters

i	Index in the sorted population

Precondition

The population must be sorted.

Definition at line 131 of file population.hh.

5.121.3.5 get_equivalent_bvs()

```
std::pair< int, int > get_equivalent_bvs (
            int index )
```

Get equivalent bit vectors.

This member function returns a pair of ints (a, b) such that,

- for all i in [0, a), f(get_best_bv(i)) $>$ f(get_best_bv(index))
- for all i in [a, b), f(get_best_bv(i)) = f(get_best_bv(index))
- for all i in [b, size), f(get_best_bv(i)) $<$ f(get_best_bv(index))

Put another way, the range [a, b) is the equivalence class of index, where two indices i and j are equivalent if f(get_best_bv(i)) = f(get_best_bv(j)).

Parameters

index	Bit vector's index in the sorted population

Precondition

The population must be sorted.

Definition at line 77 of file population.cc.

5.121.3.6 get_worst_bv()

```
bit_vector_t & get_worst_bv (
            int i )  [inline]
```

Get worst bit vector.

Parameters

i	Reversed index in the sorted population

Precondition

The population must be sorted.

Definition at line 107 of file population.hh.

5.121.3.7 partial_sort()

```
void partial_sort (
            int selection_size )  [inline]
```

Partially sort the population.

Only the permutation is sorted using the order defined by i < j if values[i] > values[j]. Before sorting, the permutation is shuffled to break ties randomly.

Parameters

selection_size	Sort the best selection_size individuals

Definition at line 164 of file population.hh.

5.121.3.8 sort()

```
void sort ( )  [inline]
```

Sort the population.

Only the permutation is sorted using the order defined by i < j if values[i] > values[j]. Before sorting, the permutation is shuffled to break ties randomly.

Definition at line 152 of file population.hh.

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

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

5.122 Population Struct Reference

Population

`#include <hnco/multiobjective/algorithms/population.hh>`

Public Types

- using **Function** = hnco::multiobjective::function::Function

  Function type
- using **value_t** = hnco::multiobjective::function::value_t

  Value type.

Public Member Functions

- **Population** ()=default
  Default constructor.
- Population (int population_size, int bv_size, int num_objectives)
  Constructor.
- int **get_size** () const
  Get the population size.
- void resize (int population_size, int bv_size, int num_objectives)
  Resize the population.
- void shrink (int population_size)
  Shrink the population.
- void **random** ()
  Sample a random population.
- void **evaluate** (Function ∗function)
  Evaluate a population.
- void **evaluate_in_parallel** (const std::vector< Function ∗ > &functions)
  Evaluate a population in parallel.

Public Attributes

- std::vector< bit_vector_t > **bvs**
  Bit vectors.
- std::vector< value_t > **values**
  Values.

5.122.1 Detailed Description

Population

Definition at line 36 of file population.hh.

5.122.2 Constructor & Destructor Documentation

5.122.2.1 Population()

```
Population (
            int population_size,
            int bv_size,
            int num_objectives ) [inline]
```

Constructor.

Parameters

population_size	Population size
bv_size	Size of bit vectors
num_objectives	Number of objectives

Definition at line 59 of file population.hh.

5.122.3 Member Function Documentation

5.122.3.1 resize()

```
void resize (
            int population_size,
            int bv_size,
            int num_objectives )  [inline]
```

Resize the population.

Parameters

population_size	Population size
bv_size	Size of bit vectors
num_objectives	Number of objectives

Definition at line 80 of file population.hh.

5.122.3.2 shrink()

```
void shrink (
            int population_size )  [inline]
```

Shrink the population.

If population_size $>$ get_size(), does nothing.

Parameters

population_size	Population size

Precondition

population_size $>$ 0

Definition at line 100 of file population.hh.

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

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

5.123 DyadicIntegerRepresentation< T >::Precision Struct Reference

Precision

`#include <hnco/representations/integer.hh>`

Public Member Functions

- **Precision** (int precision)
  Constructor.

Public Attributes

- int **precision**
  Precison.

5.123.1 Detailed Description

template<class T>
struct hnco::representation::DyadicIntegerRepresentation< T >::Precision

Precision

Definition at line 103 of file integer.hh.

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

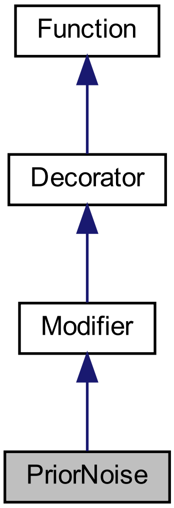
- lib/hnco/representations/integer.hh

5.124 PriorNoise Class Reference

Prior noise.

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

Inheritance diagram for PriorNoise:



Public Member Functions

- **PriorNoise** (Function ∗fn, neighborhood::Neighborhood ∗nh)

    Constructor.

Information about the function

- int **get_bv_size** () const override

    Get bit vector size.
- double get_maximum () const override

    Get the global maximum.
- bool has_known_maximum () const override

    Check for a known maximum.
- bool provides_incremental_evaluation () const override

    Check whether the function provides incremental evaluation.

Evaluation

- double **evaluate** (const bit_vector_t &) override

    Evaluate a bit vector.

Private Attributes

- neighborhood::Neighborhood ∗ **_neighborhood**

    Neighborhood.
- bit_vector_t **_noisy_bv**

    Noisy bit vector.

Additional Inherited Members

5.124.1 Detailed Description

Prior noise.

Definition at line 37 of file prior-noise.hh.

5.124.2 Member Function Documentation

5.124.2.1 get_maximum()

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

Get the global maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 69 of file prior-noise.hh.

5.124.2.2 has_known_maximum()

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

Check for a known maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 75 of file prior-noise.hh.

5.124.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Function.

Definition at line 79 of file prior-noise.hh.

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

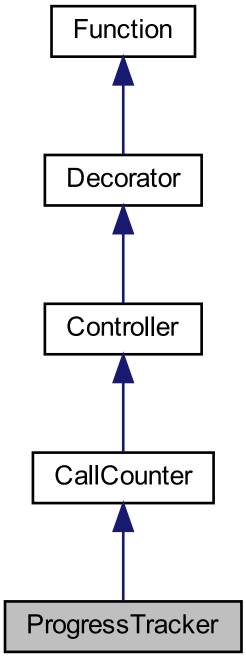
- lib/hnco/functions/modifiers/prior-noise.hh
- lib/hnco/functions/modifiers/prior-noise.cc

5.125 ProgressTracker Class Reference

Progress tracker.

```
#include <hnco/functions/controllers/controller.hh>
```

Inheritance diagram for ProgressTracker:



Classes

- struct Event
  Event

Public Member Functions

- **ProgressTracker** (Function ∗function)
  Constructor.

Evaluation

- double **evaluate** (const bit_vector_t &)
  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &bv, double value, const hnco::sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.

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

Get information

- const Event & get_last_improvement ()
  Get the last improvement.
- double **get_evaluation_time** ()
  Get evaluation time.

Setters

- void **set_log_improvement** (bool b)
  Log improvement.
- void **set_record_evaluation_time** (bool b)
  Record evaluation time.
- void **set_record_bit_vector** (bool b)
  Record bit vector.

Protected Member Functions

- void **update_last_improvement** (const bit_vector_t &bv, double value)
  Update last improvement.
- void **update_last_improvement_details** (const bit_vector_t &bv, double value)
  Update last improvement (details)

Protected Attributes

- Event **_last_improvement**
  Last improvement.
- StopWatch **_stop_watch**
  Stop watch.

Parameters

- bool **_log_improvement** = false
  Log improvement.
- bool _record_evaluation_time = false
  Record evaluation time.
- bool **_record_bit_vector** = false
  Record bit vector.

5.125.1 Detailed Description

Progress tracker.

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

Definition at line 241 of file controller.hh.

5.125.2 Member Function Documentation

5.125.2.1 get_last_improvement()

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

Get the last improvement.

Warning

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

Definition at line 331 of file controller.hh.

5.125.3 Member Data Documentation

5.125.3.1 _record_evaluation_time

```
bool _record_evaluation_time = false  [protected]
```

Record evaluation time.

Only relevant for ProgressTracker::evaluate.

Definition at line 276 of file controller.hh.

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

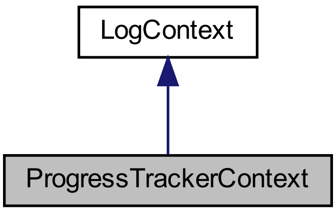
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.126 ProgressTrackerContext Class Reference

Log context for ProgressTracker.

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

Inheritance diagram for ProgressTrackerContext:



Public Member Functions

- **ProgressTrackerContext** (function::controller::ProgressTracker ∗pt)

  Constructor.
- std::string to_string ()

  Get context.

Private Attributes

- function::controller::ProgressTracker ∗ **_progress_tracker**

  Progress tracker.

5.126.1 Detailed Description

Log context for ProgressTracker.

Definition at line 49 of file log-context.hh.

5.126.2 Member Function Documentation

5.126.2.1 to_string()

```
std::string to_string ( )  [inline], [virtual]
```

Get context.

Returns

A string made of the following information:

- Number of evaluations
- Number of evaluations to find the best so far solution
- Value of the best so far solution

Implements LogContext.

Definition at line 68 of file log-context.hh.

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

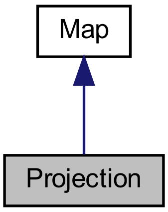
- lib/hnco/logging/log-context.hh

5.127 Projection Class Reference

Projection.

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

Inheritance diagram for Projection:



Public Member Functions

- Projection (const std::vector< int > &bit_positions, int input_size)

  Constructor.
- void **map** (const bit_vector_t &input, bit_vector_t &output) override

  Map
- int **get_input_size** () const override

  Get input size.
- int **get_output_size** () const override

  Get output size.
- bool is_surjective () const override

  Check for surjective map.

Private Attributes

- std::vector< int > **_bit_positions**

  Bit positions.
- int **_input_size**

  Input size.

5.127.1 Detailed Description

Projection.

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

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

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

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

Definition at line 549 of file map.hh.

5.127.2 Constructor & Destructor Documentation

5.127.2.1 Projection()

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

Constructor.

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

Parameters

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

Precondition

input_size >= bit_positions.size()

Definition at line 189 of file map.cc.

5.127.3 Member Function Documentation

5.127.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 587 of file map.hh.

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

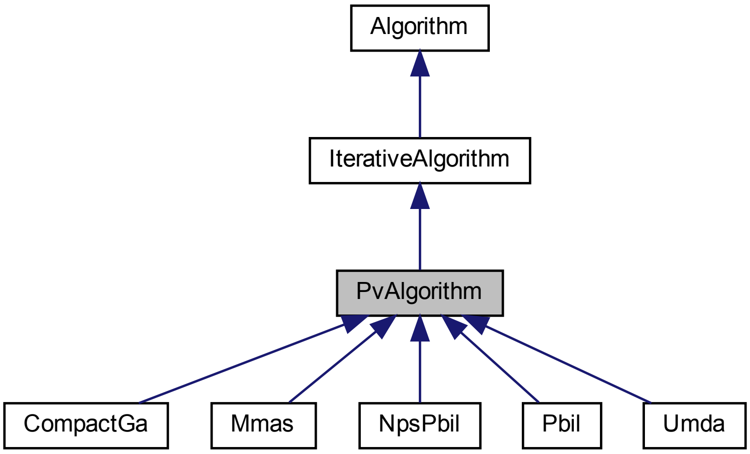
- lib/hnco/maps/map.hh
- lib/hnco/maps/map.cc

5.128 PvAlgorithm Class Reference

Probability vector algorithm.

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

Inheritance diagram for PvAlgorithm:



Public Member Functions

- **PvAlgorithm** (int n)
  Constructor.

Setters for logging

- void **set_log_entropy** (bool x)
  Log entropy.
- void **set_log_num_components** (int x)
  Set the number of probability vector components to log.
- void **set_log_pv** (bool x)
  Log probability vector.

Protected Member Functions

- void **set_something_to_log** ()
  Set flag for something to log.

Loop

- void **log** () override
  Log.

Protected Attributes

- pv_t **_pv**
  Probability vector.
- double **_lower_bound**
  Lower bound of probability.
- double **_upper_bound**
  Upper bound of probability.

Logging

- bool **_log_entropy** = false
  Log entropy.
- bool **_log_pv** = false
  Log probability vector.
- int **_log_num_components** = 5
  Number of probability vector components to log.

5.128.1 Detailed Description

Probability vector algorithm.

Definition at line 33 of file pv-algorithm.hh.

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

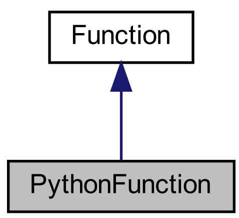
- lib/hnco/algorithms/probability-vector/pv-algorithm.hh
- lib/hnco/algorithms/probability-vector/pv-algorithm.cc

5.129 PythonFunction Class Reference

Python function.

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

Inheritance diagram for PythonFunction:



Public Member Functions

- PythonFunction (std::string path, std::string name)
  Constructor.
- ∼**PythonFunction** ()
  Destructor.
- int **get_bv_size** () const override
  Get bit vector size.
- bool **has_known_maximum** () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Private Attributes

- pybind11::object **_scope**
  Module.
- Function ∗ **_function**
  Function.

5.129.1 Detailed Description

Python function.

Uses pybind11.

The constructor initializes the python interpreter and the destructor finalizes it.

The python code must import the hnco module (built separately) to allow for communication between C++ and python. It must also define a derived class that inherits Function and an instance of it.

Definition at line 46 of file python-function.hh.

5.129.2 Constructor & Destructor Documentation

5.129.2.1 PythonFunction()

```
PythonFunction (
            std::string path,
            std::string name )
```

Constructor.

Parameters

path	Path of the python file
name	Name of the Function instance defined in the python file

Definition at line 32 of file python-function.cc.

5.129.3 Member Function Documentation

5.129.3.1 get_maximum()

```
double get_maximum ( ) const  [override], [virtual]
```

Get the global maximum.

Exceptions

std::runtime_error	

Reimplemented from Function.

Definition at line 59 of file python-function.cc.

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

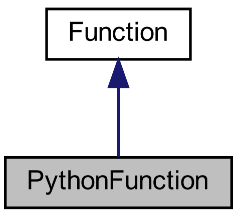
- lib/hnco/functions/collection/python-function.hh
- lib/hnco/functions/collection/python-function.cc

5.130 PythonFunction Class Reference

Python function.

```
#include <hnco/multiobjective/functions/collection/python-function.hh>
```

Inheritance diagram for PythonFunction:



Public Member Functions

- PythonFunction (std::string path, std::string name)

    Constructor.
- ∼**PythonFunction** ()

    Destructor.
- int **get_bv_size** () const

    Get bit vector size.
- int **get_output_size** () const

    Get output size (number of objectives)
- void **evaluate** (const bit_vector_t &bv, value_t &value)

    Evaluate a bit vector.

Private Attributes

- pybind11::object **_scope**

    Module.
- Function ∗ **_function**

    Function.

5.130.1 Detailed Description

Python function.

Uses pybind11.

The constructor initializes the python interpreter and the destructor finalizes it.

The python code must import the hnco module (built separately) to allow for communication between C++ and python. It must also define a derived class that inherits Function and an instance of it.

Definition at line 48 of file python-function.hh.

5.130.2 Constructor & Destructor Documentation

5.130.2.1 PythonFunction()

```
PythonFunction (
            std::string path,
            std::string name )
```

Constructor.

Parameters

path	Path of the python file
name	Name of the Function instance defined in the python file

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

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

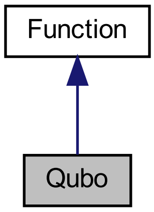
- lib/hnco/multiobjective/functions/collection/python-function.hh
- lib/hnco/multiobjective/functions/collection/python-function.cc

5.131 Qubo Class Reference

Quadratic unconstrained binary optimization.

`#include <hnco/functions/collection/qubo.hh>`

Inheritance diagram for Qubo:



Public Member Functions

- **Qubo** ()
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Load and save instance

- void load (std::string path)
  Load instance.

Private Member Functions

- void load (std::istream &stream)
  Load an instance.

Private Attributes

- std::vector< std::vector< double > > _q

  Matrix.

5.131.1 Detailed Description

Quadratic unconstrained binary optimization.

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

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

Qbsolv, a decomposing solver, finds a minimum value of a large quadratic unconstrained binary optimization (QUBO) 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.

5.131.2 Member Function Documentation

5.131.2.1 load() [1/2]

```
void load (
            std::istream & stream )   [private]
```

Load an instance.

Exceptions

std::runtime_error	

Definition at line 37 of file qubo.cc.

5.131.2.2 load() [2/2]

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 105 of file qubo.hh.

5.131.3 Member Data Documentation

5.131.3.1 _q

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

Matrix.

n x n upper triangular matrix.

Definition at line 82 of file qubo.hh.

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

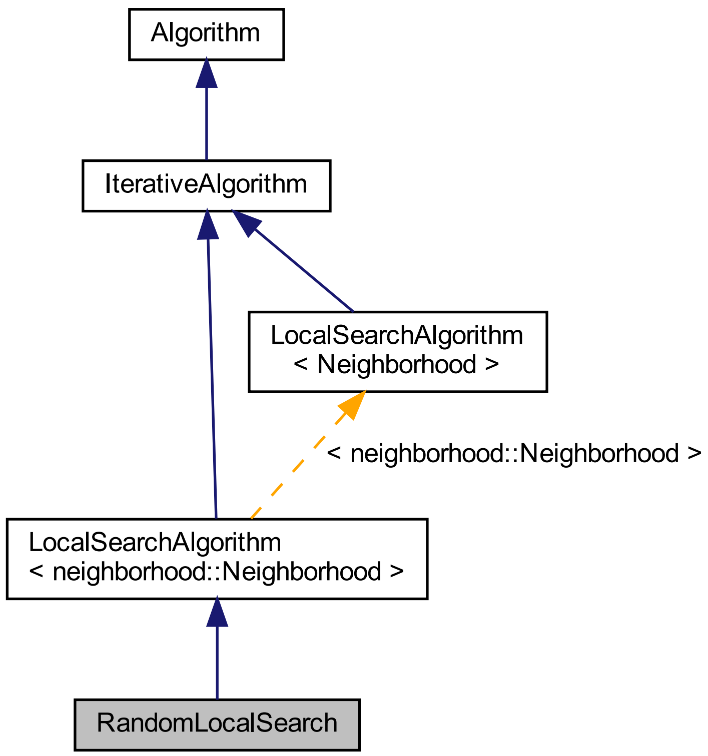
- lib/hnco/functions/collection/qubo.hh
- lib/hnco/functions/collection/qubo.cc

5.132 RandomLocalSearch Class Reference

Random local search.

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

Inheritance diagram for RandomLocalSearch:



Public Member Functions

- **RandomLocalSearch** (int n, neighborhood::Neighborhood ∗neighborhood)

  Constructor.
- void **finalize** () override

  Finalize.

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_full** ()
  Single iteration with full evaluation.
- void **iterate_incremental** ()
  Single iteration with incremental evaluation.

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- 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.

5.132.1 Detailed Description

Random local search.

Definition at line 36 of file random-local-search.hh.

5.132.2 Member Function Documentation

5.132.2.1 set_patience()

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

Set patience.

Number of consecutive rejected moves before ending the search.

Parameters

x	Patience

If x <= 0 then patience is considered infinite.

Definition at line 104 of file random-local-search.hh.

5.132.3 Member Data Documentation

5.132.3.1 _patience

```
int _patience = 50  [protected]
```

Patience.

Number of consecutive rejected moves before ending the search.

Definition at line 55 of file random-local-search.hh.

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

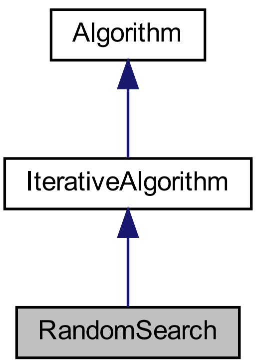
- lib/hnco/algorithms/local-search/random-local-search.hh
- lib/hnco/algorithms/local-search/random-local-search.cc

5.133 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

Loop

- void **init** () override

  Initialize.
- void **iterate** () override

  Single iteration.

Protected Attributes

- bit_vector_t **_candidate**

  Candidate.

5.133.1 Detailed Description

Random search.

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

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

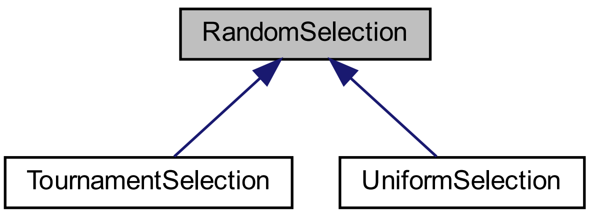
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

5.134 RandomSelection Class Reference

Random selection.

```
#include <hnco/algorithms/evolutionary-algorithms/random-selection.hh>
```

Inheritance diagram for RandomSelection:



Public Member Functions

- RandomSelection (const Population &population)

  Constructor.
- virtual void **init** ()

  Initialize.
- virtual const bit_vector_t & **select** ()=0

  Select an individual in the population.

Protected Attributes

- const Population & **_population**

  Population to select from

5.134.1 Detailed Description

Random selection.

Used as selection for reproduction in evolutionary algorithms.

Definition at line 39 of file random-selection.hh.

5.134.2 Constructor & Destructor Documentation

5.134.2.1 RandomSelection()

```
RandomSelection (
            const Population & population )   [inline]
```

Constructor.

Parameters

population	Population to select from

Definition at line 52 of file random-selection.hh.

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

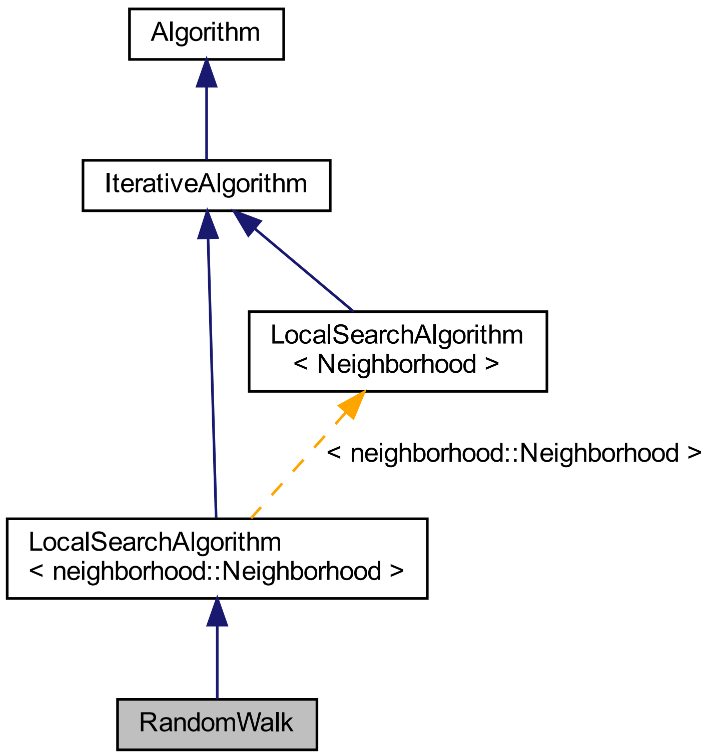
- lib/hnco/algorithms/evolutionary-algorithms/random-selection.hh

5.135 RandomWalk Class Reference

Random walk.

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

Inheritance diagram for RandomWalk:



Public Member Functions

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

    Constructor.

Setters

- void **set_incremental_evaluation** (bool x)

    Set incremental evaluation.
- void **set_log_value** ()

    Set log.

Protected Member Functions

- void **iterate_full** ()

    Single iteration with full evaluation.
- void **iterate_incremental** ()

    Single iteration with incremental evaluation.

Loop

- void **iterate** () override
  Single iteration.
- void **log** () override
  Log.

Protected Attributes

- double **_value**
  Value of the last visited bit vector.

Parameters

- bool **_incremental_evaluation** = false
  Incremental evaluation.

5.135.1 Detailed Description

Random walk.

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

Definition at line 41 of file random-walk.hh.

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

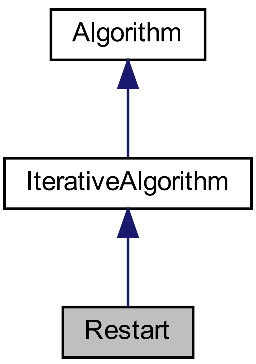
- lib/hnco/algorithms/local-search/random-walk.hh
- lib/hnco/algorithms/local-search/random-walk.cc

5.136 Restart Class Reference

Restart.

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

Inheritance diagram for Restart:



Public Member Functions

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

Protected Member Functions

Loop

- void **iterate** () override
  Single iteration.

Protected Attributes

- Algorithm ∗ **_algorithm**
  Algorithm.

5.136.1 Detailed Description

Restart.

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

Definition at line 38 of file restart.hh.

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

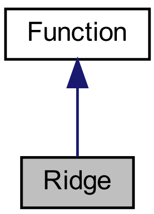
- lib/hnco/algorithms/decorators/restart.hh
- lib/hnco/algorithms/decorators/restart.cc

5.137 Ridge Class Reference

Ridge.

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

Inheritance diagram for Ridge:



Public Member Functions

- **Ridge** (int bv_size)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.

Private Attributes

- int **_bv_size**
  Bit vector size.

5.137.1 Detailed Description

Ridge.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 207 of file theory.hh.

5.137.2 Member Function Documentation

5.137.2.1 get_maximum()

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

Get the global maximum.

Returns

2 ∗ _bv_size

Reimplemented from Function.

Definition at line 230 of file theory.hh.

5.137.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 226 of file theory.hh.

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

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

5.138 ScalarToDouble< T > Struct Template Reference

Convert a scalar to a double.

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

Public Types

- using **codomain_type** = T

  Codomain type.

Public Member Functions

- double **operator()** (T x)

  Convert to double.

5.138.1 Detailed Description

template<class T>
struct hnco::function::ScalarToDouble< T >

Convert a scalar to a double.

Definition at line 32 of file converter.hh.

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

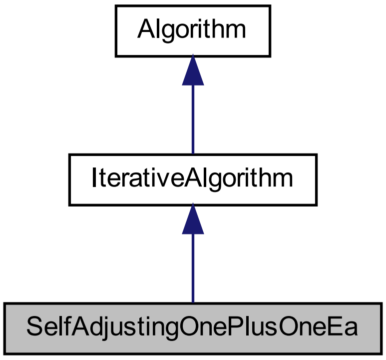
- lib/hnco/functions/converter.hh

5.139 SelfAdjustingOnePlusOneEa Class Reference

Self-adjusting (1+1) evolutionary algorithm.

`#include <hnco/algorithms/evolutionary-algorithms/self-adjusting-one-plus-one-ea.hh>`

Inheritance diagram for SelfAdjustingOnePlusOneEa:



Public Member Functions

- **SelfAdjustingOnePlusOneEa** (int n)
  Constructor.
- void **finalize** () override
  Finalize.

Setters

- void **set_mutation_rate_init** (double p)
  Set the initial mutation rate.
- void **set_mutation_rate_min** (double p)
  Set the minimum mutation rate.
- void **set_mutation_rate_max** (double p)
  Set the maximum mutation rate.
- void **set_update_strength** (double x)
  Set update strength.
- void **set_success_ratio** (double x)
  Set success ratio.
- void **set_allow_no_mutation** (bool b)
  Allow no mutation.
- void **set_incremental_evaluation** (bool b)
  Turn on incremental evaluation.

Setters for logging

- void **set_log_mutation_rate** (bool b)
  Log mutation rate.

Private Member Functions

- void **iterate_full** ()
  Single iteration with full evaluation.
- void **iterate_incremental** ()
  Single iteration with incremental evaluation.
- void **set_something_to_log** ()
  Set flag for something to log.

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.
- void **log** () override
  Log.

Private Attributes

- neighborhood::StandardBitMutation **_mutation**
  Mutation operator.
- double **_mutation_rate**
  Mutation rate.
- double **_coefficient**
  Update strength to the power the success rate.

Parameters

- double **_mutation_rate_init**
  Initial mutation rate.
- double **_mutation_rate_min**
  Minimum mutation rate.
- double **_mutation_rate_max** = 0.5
  Maximum mutation rate.
- double **_success_ratio** = 4
  Success ratio.
- double **_update_strength**
  Update strength.
- bool **_allow_no_mutation** = false
  Allow no mutation.
- bool **_incremental_evaluation** = false
  Incremental evaluation.

Logging

- bool **_log_mutation_rate** = false
  Log mutation rate.

Additional Inherited Members

5.139.1 Detailed Description

Self-adjusting (1+1) evolutionary algorithm.

Reference: Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2019. Self-adjusting mutation rates with provably optimal success rules. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19). Association for Computing Machinery, New York, NY, USA, 1479–1487. https://doi.org/10.1145/3321707.3321733

Definition at line 41 of file self-adjusting-one-plus-one-ea.hh.

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

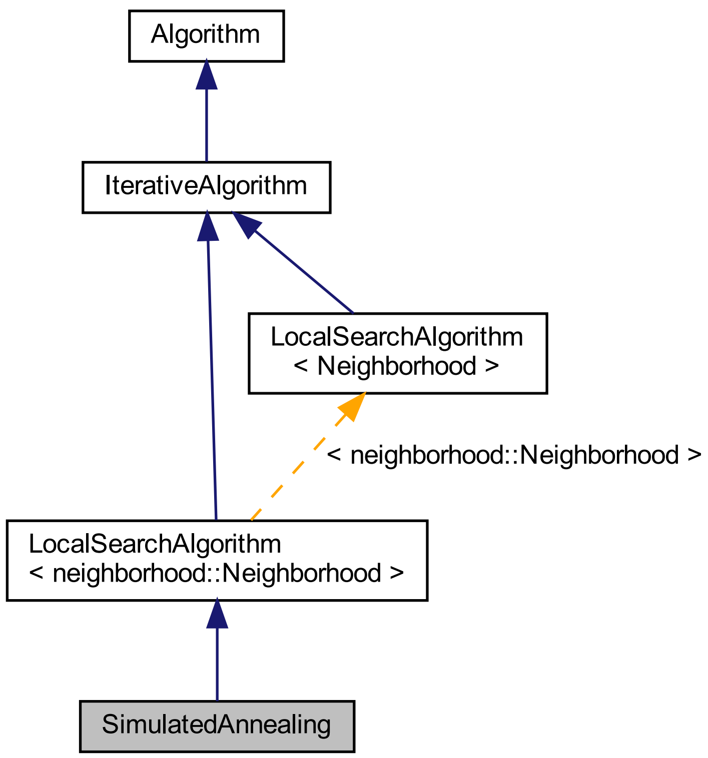
- lib/hnco/algorithms/evolutionary-algorithms/self-adjusting-one-plus-one-ea.hh
- lib/hnco/algorithms/evolutionary-algorithms/self-adjusting-one-plus-one-ea.cc

5.140 SimulatedAnnealing Class Reference

Simulated annealing.

```
#include <hnco/algorithms/local-search/simulated-annealing.hh>
```

Inheritance diagram for SimulatedAnnealing:



Public Member Functions

- **SimulatedAnnealing** (int n, neighborhood::Neighborhood ∗neighborhood)

  Constructor.

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.

Protected Member Functions

- void init_beta ()

  Initialize beta.

Loop

- void **init** () override

  Initialize.
- void **iterate** () override

  Single iteration.

Protected Attributes

- 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.

5.140.1 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 42 of file simulated-annealing.hh.

5.140.2 Member Function Documentation

5.140.2.1 init_beta()

`void init_beta ( ) [protected]`

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:

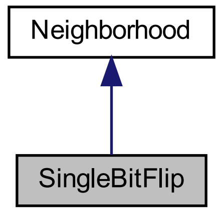
- lib/hnco/algorithms/local-search/simulated-annealing.hh
- lib/hnco/algorithms/local-search/simulated-annealing.cc

5.141 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

5.141.1 Detailed Description

One bit neighborhood.

Definition at line 163 of file neighborhood.hh.

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

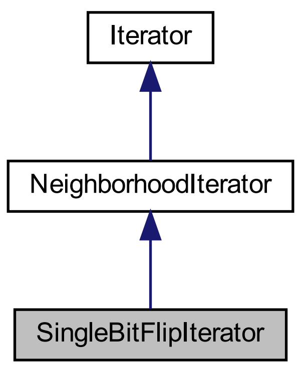
- lib/hnco/neighborhoods/neighborhood.hh

5.142 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** () override
  Has next bit vector.
- const bit_vector_t & **next** () override
  Next bit vector.

Private Attributes

- size_t **_index**
  Index of the last flipped bit.

Additional Inherited Members

5.142.1 Detailed Description

Single bit flip neighborhood iterator.

Definition at line 56 of file neighborhood-iterator.hh.

5.142.2 Constructor & Destructor Documentation

5.142.2.1 SingleBitFlipIterator()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 68 of file neighborhood-iterator.hh.

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

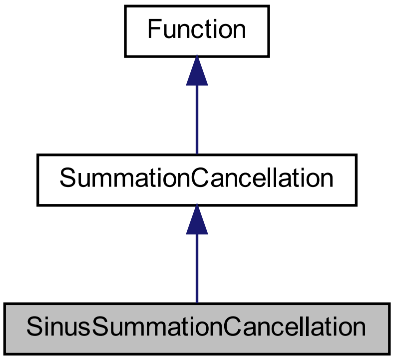
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.143 SinusSummationCancellation Class Reference

Summation cancellation with sinus.

`#include <hnco/functions/collection/cancellation.hh>`

Inheritance diagram for SinusSummationCancellation:



Public Member Functions

- **SinusSummationCancellation** (int n)

    Constructor.
- double **evaluate** (const bit_vector_t &x) override

    Evaluate a bit vector.

Additional Inherited Members

5.143.1 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 101 of file cancellation.hh.

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

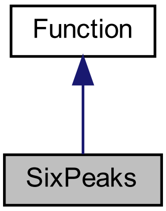
- lib/hnco/functions/collection/cancellation.hh
- lib/hnco/functions/collection/cancellation.cc

5.144 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



Public Member Functions

- **SixPeaks** (int bv_size, int threshold)
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- bool has_known_maximum () const override
  Check for a known maximum.
- double get_maximum () const override
  Get the global maximum.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Private Attributes

- int **_bv_size**
  Bit vector size.
- int **_threshold**
  Threshold.
- int **_maximum**
  Maximum.

5.144.1 Detailed Description

Six Peaks.

It is defined by

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

where:

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

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

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

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

Reference:

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

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

5.144.2 Member Function Documentation

5.144.2.1 get_maximum()

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

Get the global maximum.

Returns

2 $*$ _bv_size - _threshold - 1

Reimplemented from Function.

Definition at line 156 of file four-peaks.hh.

5.144.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 152 of file four-peaks.hh.

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

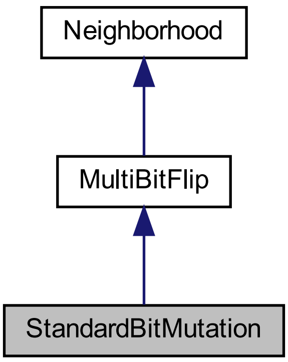
- lib/hnco/functions/collection/four-peaks.hh
- lib/hnco/functions/collection/four-peaks.cc

5.145 StandardBitMutation Class Reference

Standard bit mutation.

`#include <hnco/neighborhoods/neighborhood.hh>`

Inheritance diagram for StandardBitMutation:



Public Member Functions

- StandardBitMutation (int n)
  Constructor.
- StandardBitMutation (int n, double p)
  Constructor.

Setters

- void set_mutation_rate (double p)
  Set mutation rate.
- void **set_allow_no_mutation** (bool b)
  Set the flag _allow_no_mutation.

Private Member Functions

- void **sample_bits** ()
  Sample bits.
- void **bernoulli_process** ()
  Bernoulli process.

Private Attributes

- std::bernoulli_distribution **_bernoulli_dist**
  Bernoulli distribution (biased coin)
- std::binomial_distribution< int > **_binomial_dist**
  Binomial distribution.
- bool **_rejection_sampling** = false
  Rejection sampling.

Parameters

- bool **_allow_no_mutation** = false
  Allow no mutation.

Additional Inherited Members

5.145.1 Detailed Description

Standard bit mutation.

Each component of the origin bit vector is flipped with some fixed probability. Unless stated otherwise, 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 222 of file neighborhood.hh.

5.145.2 Constructor & Destructor Documentation

5.145.2.1 StandardBitMutation() [1/2]

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

Constructor.

Parameters

n	Size of bit vectors

The Bernoulli probability is set to 1 / n.

Definition at line 257 of file neighborhood.hh.

5.145.2.2 StandardBitMutation() [2/2]

```
StandardBitMutation (
            int n,
            double p )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors
p	Bernoulli probability

Definition at line 267 of file neighborhood.hh.

5.145.3 Member Function Documentation

5.145.3.1 set_mutation_rate()

```
void set_mutation_rate (
            double p )  [inline]
```

Set mutation rate.

Sets _rejection_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 282 of file neighborhood.hh.

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

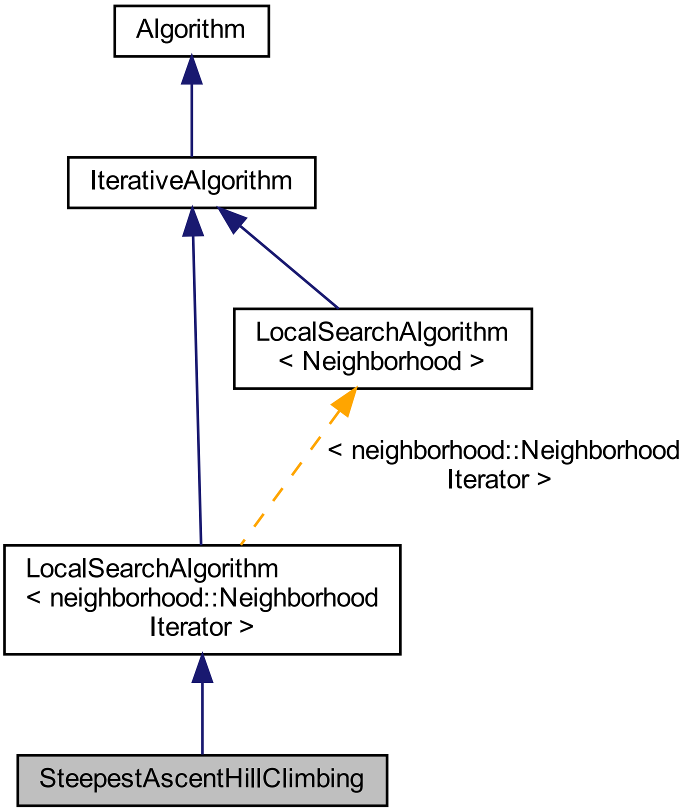
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.146 SteepestAscentHillClimbing Class Reference

Steepest ascent hill climbing.

`#include <hnco/algorithms/local-search/steepest-ascent-hill-climbing.hh>`

Inheritance diagram for SteepestAscentHillClimbing:



Public Member Functions

- **SteepestAscentHillClimbing** (int n, neighborhood::NeighborhoodIterator ∗neighborhood)

    Constructor.

Protected Member Functions

- void **iterate** () override

    Single iteration.

Protected Attributes

- std::vector< bit_vector_t > **_candidates**

  Potential candidate.

5.146.1 Detailed Description

Steepest ascent hill climbing.

Definition at line 34 of file steepest-ascent-hill-climbing.hh.

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

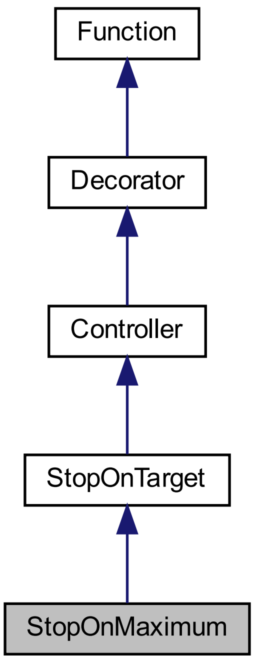
- lib/hnco/algorithms/local-search/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/local-search/steepest-ascent-hill-climbing.cc

5.147 StopOnMaximum Class Reference

Stop on maximum.

```
#include <hnco/functions/controllers/controller.hh>
```

Inheritance diagram for StopOnMaximum:



Public Member Functions

- StopOnMaximum (Function ∗function)

  Constructor.

Additional Inherited Members

5.147.1 Detailed Description

Stop on maximum.

Definition at line 144 of file controller.hh.

5.147.2 Constructor & Destructor Documentation

5.147.2.1 StopOnMaximum()

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

Constructor.

Precondition

function->has_known_maximum()

Definition at line 151 of file controller.hh.

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

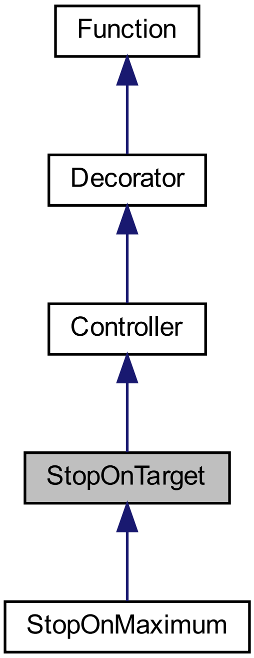
- lib/hnco/functions/controllers/controller.hh

5.148 StopOnTarget Class Reference

Stop on target.

`#include <hnco/functions/controllers/controller.hh>`

Inheritance diagram for StopOnTarget:



Public Member Functions

- StopOnTarget (Function ∗function, double target)

  Constructor.
- const algorithm::solution_t & **get_trigger** ()

  Get trigger.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &bv, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &bv, double value)

  Update after a safe evaluation.

Private Attributes

- double **_target**

    Target.
- algorithm::solution_t **_trigger**

    Trigger.

Additional Inherited Members

5.148.1 Detailed Description

Stop on target.

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

Warning

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

Definition at line 93 of file controller.hh.

5.148.2 Constructor & Destructor Documentation

5.148.2.1 StopOnTarget()

```
StopOnTarget (
            Function * function,
            double target )  [inline]
```

Constructor.

Parameters

function	Decorated function
target	Target

Definition at line 108 of file controller.hh.

5.148.3 Member Function Documentation

5.148.3.1 evaluate()

```
double evaluate (
            const bit_vector_t & bv )  [virtual]
```

Evaluate a bit vector.

Exceptions

TargetReached	

Implements Function.

Definition at line 32 of file controller.cc.

5.148.3.2 evaluate_incrementally()

```
double evaluate_incrementally (
            const bit_vector_t & bv,
            double value,
            const hnco::sparse_bit_vector_t & flipped_bits )  [virtual]
```

Incrementally evaluate a bit vector.

Exceptions

TargetReached	

Reimplemented from Function.

Definition at line 45 of file controller.cc.

5.148.3.3 update()

```
void update (
            const bit_vector_t & bv,
            double value )  [virtual]
```

Update after a safe evaluation.

Exceptions

TargetReached	

Reimplemented from Function.

Definition at line 58 of file controller.cc.

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

- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.149 StopWatch Class Reference

Stop watch.

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

Public Member Functions

- void **start** ()
  Start.
- void **stop** ()
  Stop.
- double **get_total_time** ()
  Get total time.
- void **reset** ()
  Reset.

Private Attributes

- double **_total_time** = 0
  Total time.
- clock_t **_start**
  Start time.

5.149.1 Detailed Description

Stop watch.

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

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

- lib/hnco/stop-watch.hh

5.150 Sudoku Class Reference

Sudoku

`#include <hnco/functions/collection/sudoku.hh>`

Public Types

- using **domain_type** = std::size_t

  Domain type.
- using **codomain_type** = double

  Codomain type.

Public Member Functions

- **Sudoku** ()

  Default constructor.
- void random (int c)

  Random instance.
- int **get_num_variables** ()

  Get the number of variables.
- void **display** (std::ostream &stream) const

  Display the problem.
- void **describe** (const std::vector< domain_type > &x, std::ostream &stream)

  Describe a solution.
- double **evaluate** (const std::vector< domain_type > &x)

  Evaluate a solution.

Private Member Functions

- void **write_variables** (const std::vector< domain_type > &x)

  Write variables.

Private Attributes

- std::vector< std::vector< char > > **_problem_instance**

  Problem instance.
- std::vector< std::vector< domain_type > > **_candidate**

  Candidate.
- std::vector< int > **_counts**

  Counts.
- int **_num_variables**

  Number of variables.

Load and save instance

- void load_ (std::istream &stream)

  Load an instance.
- void **save_** (std::ostream &stream) const

  Save an instance.
- void load (std::string path)

  Load instance.
- void save (std::string path) const

  Save instance.

5.150.1 Detailed Description

Sudoku

Definition at line 34 of file sudoku.hh.

5.150.2 Member Function Documentation

5.150.2.1 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 100 of file sudoku.hh.

5.150.2.2 load_()

```
void load_ (
            std::istream & stream )  [private]
```

Load an instance.

Exceptions

std::runtime_error	

Definition at line 57 of file sudoku.cc.

5.150.2.3 random()

```
void random (
            int c )
```

Random instance.

Parameters

c	Number of empty cells

Definition at line 96 of file sudoku.cc.

5.150.2.4 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 112 of file sudoku.hh.

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

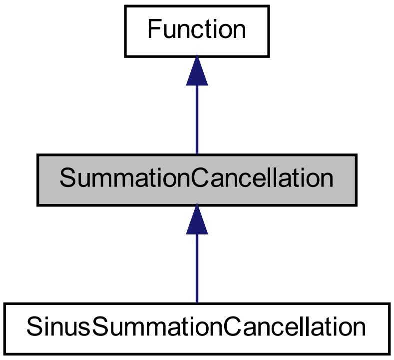
- lib/hnco/functions/collection/sudoku.hh
- lib/hnco/functions/collection/sudoku.cc

5.151 SummationCancellation Class Reference

Summation cancellation.

```
#include <hnco/functions/collection/cancellation.hh>
```

Inheritance diagram for SummationCancellation:



Public Member Functions

- SummationCancellation (int n)

  Constructor.
- int **get_bv_size** () const override

  Get bit vector size.
- bool has_known_maximum () const override

  Check for a known maximum.
- double **get_maximum** () const override

  Get the global maximum.
- double **evaluate** (const bit_vector_t &x) override

  Evaluate a bit vector.

Protected Member Functions

- void **convert** (const bit_vector_t &x)

  Convert a bit vector into a real vector.

Protected Attributes

- int **_bv_size**

  Bit vector size.
- std::vector< double > **_buffer**

  Buffer.

5.151.1 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 46 of file cancellation.hh.

5.151.2 Constructor & Destructor Documentation

5.151.2.1 SummationCancellation()

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

Constructor.

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

Parameters

n	Size of the bit vector

Definition at line 68 of file cancellation.hh.

5.151.3 Member Function Documentation

5.151.3.1 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 81 of file cancellation.hh.

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

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

5.152 SymmetricWalshMoment2 Struct Reference

Symmetric Walsh moment.

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

Public Member Functions

- SymmetricWalshMoment2 (int n)

  Constructor.
- void display (std::ostream &stream)

  Display Walsh moment.
- void **init** ()

  Initialize Walsh moment.
- void **add** (const bit_vector_t &bv)

  Add a bit vector to a Walsh moment.
- void average (int count)

  Average each Walsh moment.
- void update (const SymmetricWalshMoment2 &wm, double rate)

  Update a Walsh moment.
- void update (const SymmetricWalshMoment2 &wm1, const SymmetricWalshMoment2 &wm2, double rate)

  Update a Walsh moment.
- void scaled_difference (double lambda, const SymmetricWalshMoment2 &wm1, const SymmetricWalshMoment2 &wm2)

  Compute a scaled difference between two moments.
- void bound (double margin)

  Bound Walsh moment.
- double **norm_1** () const

  1-norm of the Walsh moment
- double **norm_2** () const

  2-norm of the Walsh moment
- double **norm_infinite** () const

  infinite-norm of the Walsh moment
- double **distance** (const SymmetricWalshMoment2 &wm) const

  distance between the Walsh moment and another Walsh moment

Public Attributes

- std::vector< double > **first_moment**

  First moment.
- std::vector< std::vector< double > > **second_moment**

  Second moment.

5.152.1 Detailed Description

Symmetric Walsh moment.

Definition at line 144 of file walsh-moment.hh.

5.152.2 Constructor & Destructor Documentation

5.152.2.1 SymmetricWalshMoment2()

```
SymmetricWalshMoment2 (
            int n )
```

Constructor.

Parameters

n	Size of bit vector

Definition at line 237 of file walsh-moment.cc.

5.152.3 Member Function Documentation

5.152.3.1 average()

```
void average (
            int count )
```

Average each Walsh moment.

Postcondition

matrix_is_symmetric(second_moment)

Definition at line 297 of file walsh-moment.cc.

5.152.3.2 bound()

```
void bound (
            double margin )
```

Bound Walsh moment.

Ensure that the distance from each Walsh moment to the -1/1 bounds is greater or equal to the given margin.

Parameters

margin	Distance from the -1/1 bounds

Definition at line 379 of file walsh-moment.cc.

5.152.3.3 display()

```
void display (
            std::ostream & stream )
```

Display Walsh moment.

A SymmetricWalshMoment2 is displayed as a full symmetric matrix with diagonal entries equal to first moments and off-diagonal entries equal to second moments.

Definition at line 248 of file walsh-moment.cc.

5.152.3.4 scaled_difference()

```
void scaled_difference (
            double lambda,
            const SymmetricWalshMoment2 & wm1,
            const SymmetricWalshMoment2 & wm2 )
```

Compute a scaled difference between two moments.

This member function implements:

self = lambda $*$ wm1 - wm2

It is mostly useful in herding (Hea).

Parameters

lambda	Scale
wm1	First Walsh moment
wm2	Second Walsh moment

Definition at line 358 of file walsh-moment.cc.

5.152.3.5 update() [1/2]

```
void update (
            const SymmetricWalshMoment2 & wm,
            double rate )
```

Update a Walsh moment.

This member function implements:

self += rate ∗ (wm1 - self)

Parameters

wm	Target Walsh moment
rate	Learning rate

Postcondition

For all i, is_in_interval(first_moment[i], -1, 1)

For all i != j, is_in_interval(second_moment[i][j], -1, 1)

matrix_is_symmetric(second_moment)

Definition at line 315 of file walsh-moment.cc.

5.152.3.6 update() [2/2]

```
void update (
            const SymmetricWalshMoment2 & wm1,
            const SymmetricWalshMoment2 & wm2,
            double rate )
```

Update a Walsh moment.

This member function implements:

self += rate ∗ (wm1 - wm2)

The resulting entries are not necessarily those of a Walsh moment, that is

is_in_interval(first_moment[i], -1, 1) or

is_in_interval(second_moment[i][j], -1, 1)

might fail for some i != j.

Parameters

wm1	Target Walsh moment
wm2	Walsh moment to move away from
rate	Learning rate

Definition at line 336 of file walsh-moment.cc.

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

- lib/hnco/algorithms/walsh-moment/walsh-moment.hh
- lib/hnco/algorithms/walsh-moment/walsh-moment.cc

5.153 SymmetricWalshMoment2GibbsSampler Class Reference

Gibbs sampler with symmetric Walsh moments.

```
#include <hnco/algorithms/walsh-moment/gibbs-sampler.hh>
```

Public Types

- using **Moment** = SymmetricWalshMoment2

    Walsh moment type.

Public Member Functions

- **SymmetricWalshMoment2GibbsSampler** (int n, const SymmetricWalshMoment2 &mp)

    Constructor.
- void **init** ()

    Initialize.
- void **update** (int i)

    Update state.
- void **update_sync** ()

    Update state synchronously.
- const bit_vector_t & **get_state** ()

    Get the state of the Gibbs sampler.

Private Attributes

- const SymmetricWalshMoment2 & **_model_parameters**

    Model parameters.
- bit_vector_t **_state**

    State of the Gibbs sampler.
- pv_t **_pv**

    Probability vector for synchronous Gibbs sampling.

5.153.1 Detailed Description

Gibbs sampler with symmetric Walsh moments.

Definition at line 75 of file gibbs-sampler.hh.

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

- lib/hnco/algorithms/walsh-moment/gibbs-sampler.hh
- lib/hnco/algorithms/walsh-moment/gibbs-sampler.cc

5.154 SymmetricWalshMoment2Herding Class Reference

Herding with symmetric Walsh moment.

```
#include <hnco/algorithms/walsh-moment/herding.hh>
```

Public Types

- using **Moment** = SymmetricWalshMoment2

    Walsh moment type.

Public Member Functions

- SymmetricWalshMoment2Herding (int n)

    Constructor.
- void **init** ()

    Initialization.
- void **sample** (const SymmetricWalshMoment2 &target, bit_vector_t &x)

    Sample a bit vector.
- double **error** (const SymmetricWalshMoment2 &target)

    Compute the error.

Getters

- const SymmetricWalshMoment2 & **get_delta** () const

    Get delta.

Setters

- void **set_randomize_bit_order** (bool x)

    Randomize bit order.

Protected Attributes

- SymmetricWalshMoment2 **_delta**

  Delta moment.
- SymmetricWalshMoment2 **_count**

  Counter moment.
- SymmetricWalshMoment2 **_error**

  Error moment.
- permutation_t **_permutation**

  Permutation.
- int **_time**

  Time.

Parameters

- bool **_randomize_bit_order** = false

  Randomize bit order.

5.154.1 Detailed Description

Herding with symmetric Walsh moment.

Definition at line 112 of file herding.hh.

5.154.2 Constructor & Destructor Documentation

5.154.2.1 SymmetricWalshMoment2Herding()

```
SymmetricWalshMoment2Herding (
            int n )  [inline]
```

Constructor.

Parameters

n	Size of bit vectors

Definition at line 149 of file herding.hh.

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

- lib/hnco/algorithms/walsh-moment/herding.hh
- lib/hnco/algorithms/walsh-moment/herding.cc

5.155 TargetReached Class Reference

Target reached.

`#include <hnco/exception.hh>`

Inherits runtime_error.

5.155.1 Detailed Description

Target reached.

Definition at line 40 of file exception.hh.

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

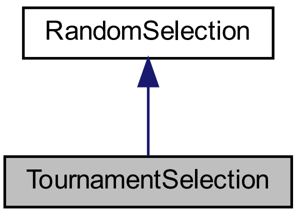
- lib/hnco/exception.hh

5.156 TournamentSelection Class Reference

Tournament selection.

`#include <hnco/algorithms/evolutionary-algorithms/random-selection.hh>`

Inheritance diagram for TournamentSelection:



Public Member Functions

- TournamentSelection (const Population &population)
  Constructor.
- void **init** () override
  Initialize.
- const bit_vector_t & select () override
  Select an individual in the population.

Setters

- void **set_tournament_size** (int n)
  Set the tournament size.

Private Attributes

- hnco::multiobjective::algorithm::TournamentSelection< double, std::greater< double > > **_tournament_↩ selection**

    Tournament selection.

Parameters

- int **_tournament_size** = 2

    Tournament size.

Additional Inherited Members

5.156.1 Detailed Description

Tournament selection.

Reuses the hnco::multiobjective::algorithm::TournamentSelection class.

Definition at line 93 of file random-selection.hh.

5.156.2 Constructor & Destructor Documentation

5.156.2.1 TournamentSelection()

```
TournamentSelection (
            const Population & population )  [inline]
```

Constructor.

Parameters

population	Population to select from

Definition at line 115 of file random-selection.hh.

5.156.3 Member Function Documentation

5.156.3.1 select()

```
const bit_vector_t & select ( )  [override], [virtual]
```

Select an individual in the population.

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

Precondition

The population must be evaluated.

Implements RandomSelection.

Definition at line 45 of file random-selection.cc.

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

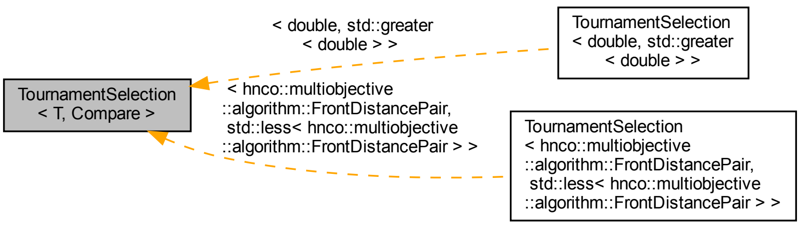
- lib/hnco/algorithms/evolutionary-algorithms/random-selection.hh
- lib/hnco/algorithms/evolutionary-algorithms/random-selection.cc

5.157 TournamentSelection< T, Compare > Class Template Reference

Tournament selection.

`#include <hnco/multiobjective/algorithms/random-selection.hh>`

Inheritance diagram for TournamentSelection< T, Compare >:



Public Member Functions

- **TournamentSelection** (const std::vector< bit_vector_t > &bvs, const std::vector< T > &values)

  Constructor.
- void **init** ()

  Initialize.
- const bit_vector_t & **select** ()

  Select a bit vector.

Setters

- void **set_tournament_size** (int n)

  Set the tournament size.

Private Attributes

- const std::vector< bit_vector_t > & **_bvs**

  Bit vectors.
- const std::vector< T > & **_values**

  Values.
- hnco::permutation_t **_permutation**

  Permutation.
- int **_start**

  Beginning of the slice of permutation used in a tournament round.
- int **_stop**

  End of the slice of permutation used in a tournament round.
- Compare **_compare**

  Comparison operator.

Parameters

- int **_tournament_size** = 2

  Tournament size.

5.157.1 Detailed Description

template<typename T, typename Compare>
class hnco::multiobjective::algorithm::TournamentSelection< T, Compare >

Tournament selection.

Implement tournament selection without replacement as explained in the reference:

Goldberg, Korb, and Deb, "Messy genetic algorithms: Motivation, analysis, and first results", Complex systems, 1989.

`https://www.complex-systems.com/abstracts/v03_i05_a05/`

Definition at line 45 of file random-selection.hh.

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

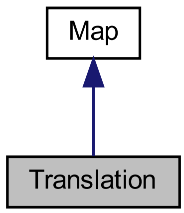
- lib/hnco/multiobjective/algorithms/random-selection.hh

5.158 Translation Class Reference

Translation.

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

Inheritance diagram for Translation:



Public Member Functions

- void **map** (const bit_vector_t &input, bit_vector_t &output) override
  Map
- int **get_input_size** () const override
  Get input size.
- int **get_output_size** () const override
  Get output size.
- bool is_surjective () const override
  Check for surjective map.
- void **display** (std::ostream &stream) const override
  Display.
- void **random** (int n)
  Random instance.
- void **set_bv** (const bit_vector_t &bv)
  Set the translation vector.

Load and save map

- void load (std::string path)
  Load map.
- void save (std::string path) const
  Save map.

Private Member Functions

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

Private Attributes

- bit_vector_t **_bv**

  Translation vector

5.158.1 Detailed Description

Translation.

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

Definition at line 80 of file map.hh.

5.158.2 Member Function Documentation

5.158.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 122 of file map.hh.

5.158.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 147 of file map.hh.

5.158.2.3 save()

```
void save (
            std::string path ) const  [inline]
```

Save map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 154 of file map.hh.

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

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

5.159 Transvection Struct Reference

Transvection.

`#include <hnco/maps/transvection.hh>`

Public 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.
- bool **is_valid** () const
  Check validity.
- bool is_valid (int n) const
  Check validity.
- void **display** (std::ostream &stream) const
  Display transvection.
- void random (int n)

Sample a random transvection.
- void random_non_commuting (int n, const Transvection &a)

  Sample a random transvection.
- void multiply (bit_vector_t &x) const

  Multiply a bit vector from the left.
- void multiply (bit_matrix_t &M) const

  Multiply a bit matrix from the left.

Public Attributes

- int **row_index**

  Row index.
- int **column_index**

  Column index.

5.159.1 Detailed Description

Transvection.

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

Transvections generate invertible matrices over the finite field F_2.

Definition at line 63 of file transvection.hh.

5.159.2 Member Function Documentation

5.159.2.1 is_valid()

```
bool is_valid (
              int n ) const
```

Check validity.

Parameters

n	Dimension

Definition at line 48 of file transvection.cc.

5.159.2.2 multiply() [1/2]

```
void multiply (
```

```
            bit_matrix_t & M ) const
```

Multiply a bit matrix from the left.

Parameters

M	Bit matrix

Precondition

is_valid()

is_valid(bm_num_rows(M))

Warning

This function modifies the given bit vector.

Definition at line 117 of file transvection.cc.

5.159.2.3 multiply() [2/2]

```
void multiply (
            bit_vector_t & x ) const
```

Multiply a bit vector from the left.

Parameters

x	Bit vector

Precondition

is_valid()

is_valid(x.size())

Warning

This function modifies the given bit vector.

Definition at line 105 of file transvection.cc.

5.159.2.4 random()

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

Sample a random transvection.

Parameters

n	Dimension

Precondition

n > 1

Definition at line 61 of file transvection.cc.

5.159.2.5 random_non_commuting()

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

Sample a random transvection.

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

Parameters

n	Dimension
a	Given transvection

Precondition

n > 1

Definition at line 77 of file transvection.cc.

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

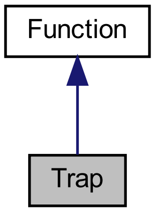
- lib/hnco/maps/transvection.hh
- lib/hnco/maps/transvection.cc

5.160 Trap Class Reference

Trap.

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

Inheritance diagram for Trap:



Public Member Functions

- Trap (int bv_size, int num_traps)

  Constructor.
- int **get_bv_size** () const

  Get bit vector size.
- double **evaluate** (const bit_vector_t &)

  Evaluate a bit vector.
- bool has_known_maximum () const

  Check for a known maximum.
- double get_maximum () const

  Get the global maximum.

Private Attributes

- int **_bv_size**

  Bit vector size.
- int **_num_traps**

  Number of traps.
- int **_trap_size**

  Trap size

5.160.1 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.

5.160.2 Constructor & Destructor Documentation

5.160.2.1 Trap()

```
Trap (
            int bv_size,
            int num_traps )  [inline]
```

Constructor.

Parameters

bv_size	Bit vector size
num_traps	Number of traps

Warning

bv_size must be a multiple of num_traps

Definition at line 64 of file trap.hh.

5.160.3 Member Function Documentation

5.160.3.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 88 of file trap.hh.

5.160.3.2 has_known_maximum()

```
bool has_known_maximum ( ) const  [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:

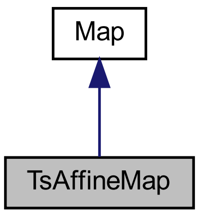
- lib/hnco/functions/collection/trap.hh
- lib/hnco/functions/collection/trap.cc

5.161 TsAffineMap Class Reference

Transvection sequence affine map.

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

Inheritance diagram for TsAffineMap:



Public Types

- enum class SamplingMode {
  unconstrained , commuting_transvections , unique_source , unique_destination ,
  disjoint_transvections , non_commuting_transvections }

  Sampling mode.

Public Member Functions

- void random (int n, int t, SamplingMode mode)
  Random instance.
- void **map** (const bit_vector_t &input, bit_vector_t &output) override
  Map
- int **get_input_size** () const override
  Get input size.
- int **get_output_size** () const override
  Get output size.
- bool is_surjective () const override
  Check for surjective map.
- void **display** (std::ostream &stream) const override
  Display.
- void **inverse** ()
  Inverse.

Load and save map

- void load (std::string path)
  Load map.
- void save (std::string path) const
  Save 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

- transvection_sequence_t **_ts**
  Transvection sequence
- bit_vector_t **_bv**
  Translation vector

5.161.1 Detailed Description

Transvection sequence affine map.

An affine map f from F_2^m to F_2^n is defined by $f(x) = Ax+b$, where A is an n x m bit matrix and b is an n-dimensional bit vector.

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

Definition at line 601 of file map.hh.

5.161.2 Member Enumeration Documentation

5.161.2.1 SamplingMode

```
enum class SamplingMode  [strong]
```

Sampling mode.

Enumerator

unconstrained	Unconstrained.
commuting_transvections	Commuting transvections.
unique_source	Transvection sequence with unique source
unique_destination	Transvection sequence with unique destination
disjoint_transvections	Disjoint transvections.
non_commuting_transvections	Non commuting transvections.

Definition at line 637 of file map.hh.

5.161.3 Member Function Documentation

5.161.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 680 of file map.hh.

5.161.3.2 load()

```
void load (
            std::string path )  [inline]
```

Load map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 697 of file map.hh.

5.161.3.3 random()

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

Random instance.

Parameters

n	Dimension
t	Length of sequence of transvections
mode	Sampling mode

Definition at line 210 of file map.cc.

5.161.3.4 save()

```
void save (
            std::string path ) const [inline]
```

Save map.

Parameters

path	Path of the file

Exceptions

std::runtime_error	

Definition at line 704 of file map.hh.

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

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

5.162 Tsp Class Reference

Traveling salesman problem.

`#include <hnco/functions/collection/tsp.hh>`

Public Member Functions

- **Tsp** ()
  Default constructor.
- int **get_num_elements** () const
  Get the number of elements.
- void **display** (std::ostream &stream) const
  Display the problem.
- void **describe** (const hnco::permutation_t &permutation, std::ostream &stream)
  Describe a solution.
- double **evaluate** (const hnco::permutation_t &permutation)
  Evaluate a solution.

Instance generators

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

Private Attributes

- std::vector< std::vector< float > > **_distances**
  Distances.

Load and save instance

- void load_ (std::istream &stream)
  Load an instance.
- void **load_coordinates** (std::istream &stream)
- void **save_** (std::ostream &stream) const
  Save an instance.
- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

5.162.1 Detailed Description

Traveling salesman problem.

Source: TSPLIB 95, Gerhard Reinelt

Definition at line 40 of file tsp.hh.

5.162.2 Member Function Documentation

5.162.2.1 generate()

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

Instance generator.

Parameters

n	Number of vertices
generator	Generator for distances

Definition at line 94 of file tsp.hh.

5.162.2.2 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 129 of file tsp.hh.

5.162.2.3 load_()

```
void load_ (
             std::istream & stream )  [private]
```

Load an instance.

Exceptions

std::runtime_error	

Definition at line 32 of file tsp.cc.

5.162.2.4 random()

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

Random instance.

Distances are sampled from the normal distribution.

Parameters

n	Number of vertices

Definition at line 113 of file tsp.hh.

5.162.2.5 save()

```
void save (
             std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 141 of file tsp.hh.

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

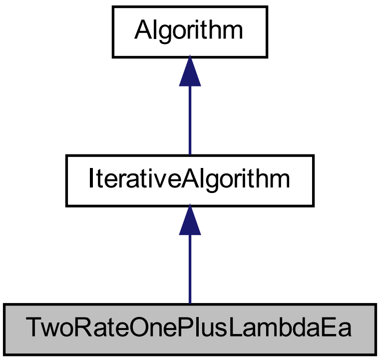
- lib/hnco/functions/collection/tsp.hh
- lib/hnco/functions/collection/tsp.cc

5.163 TwoRateOnePlusLambdaEa Class Reference

Two-rate (1+lambda) evolutionary algorithm.

```
#include <hnco/algorithms/evolutionary-algorithms/two-rate-one-plus-lambda-ea.↵
hh>
```

Inheritance diagram for TwoRateOnePlusLambdaEa:



Public Member Functions

- **TwoRateOnePlusLambdaEa** (int n, int population_size)
  Constructor.

Setters

- void **set_mutation_rate_init** (double r)
  Set the initial mutation rate.
- void **set_allow_no_mutation** (bool b)
  Allow no mutation.

Setters for logging

- void **set_log_mutation_rate** (bool b)
  Log mutation rate.

Protected Member Functions

- void **set_something_to_log** ()
  Set flag for something to log.

Loop

- void **init** () override
  Initialization.
- void **iterate** () override
  Single iteration.
- void **log** () override
  Log.

Protected Attributes

- Population **_population**
  Population.
- neighborhood::StandardBitMutation **_mutation_operator**
  Mutation operator.
- double **_mutation_rate**
  Mutation rate.

Parameters

- double **_mutation_rate_init**
  Initial mutation rate.
- bool **_allow_no_mutation** = false
  Allow no mutation.

Logging

- bool **_log_mutation_rate** = false
  Log entropy.

5.163.1 Detailed Description

Two-rate (1+lambda) evolutionary algorithm.

Reference:

Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang.

1. The (1+lambda) evolutionary algorithm with self-adjusting mutation rate. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17). Association for Computing Machinery, New York, NY, USA, 1351–1358. `https://doi.org/10.1145/3071178.3071279`

Definition at line 47 of file two-rate-one-plus-lambda-ea.hh.

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

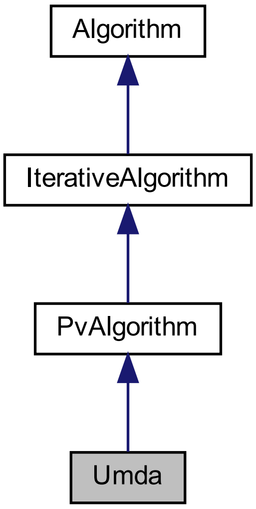
- lib/hnco/algorithms/evolutionary-algorithms/two-rate-one-plus-lambda-ea.hh
- lib/hnco/algorithms/evolutionary-algorithms/two-rate-one-plus-lambda-ea.cc

5.164 Umda Class Reference

Univariate marginal distribution algorithm.

```
#include <hnco/algorithms/probability-vector/umda.hh>
```

Inheritance diagram for Umda:



Public Member Functions

- **Umda** (int n, int population_size)
  Constructor.

Setters

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

Protected Member Functions

Loop

- void **init** () override
  Initialize.
- void **iterate** () override
  Single iteration.

Protected Attributes

- Population **_population**

  Population.

Parameters

- int **_selection_size** = 1

  Selection size.

5.164.1 Detailed Description

Univariate marginal distribution algorithm.

Reference:

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

Definition at line 41 of file umda.hh.

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

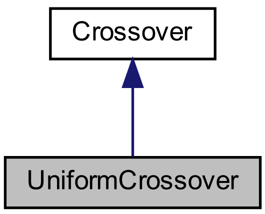
- lib/hnco/algorithms/probability-vector/umda.hh
- lib/hnco/algorithms/probability-vector/umda.cc

5.165 UniformCrossover Class Reference

Uniform crossover.

`#include <hnco/algorithms/evolutionary-algorithms/crossover.hh>`

Inheritance diagram for UniformCrossover:



Public Member Functions

- void recombine (const bit_vector_t &parent1, const bit_vector_t &parent2, bit_vector_t &offspring)
  Recombine.

5.165.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

5.165.2 Member Function Documentation

5.165.2.1 recombine()

```
void recombine (
            const bit_vector_t & parent1,
            const bit_vector_t & parent2,
            bit_vector_t & offspring )  [virtual]
```

Recombine.

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:

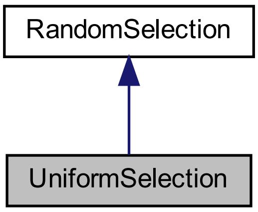
- lib/hnco/algorithms/evolutionary-algorithms/crossover.hh
- lib/hnco/algorithms/evolutionary-algorithms/crossover.cc

5.166 UniformSelection Class Reference

Uniform selection.

```
#include <hnco/algorithms/evolutionary-algorithms/random-selection.hh>
```

Inheritance diagram for UniformSelection:



Public Member Functions

- UniformSelection (const Population &population)

  Constructor.
- const bit_vector_t & **select** () override

  Select an individual in the population.

Private Attributes

- std::uniform_int_distribution< int > **_choose_individual**

  Random index.

Additional Inherited Members

5.166.1 Detailed Description

Uniform selection.

Definition at line 66 of file random-selection.hh.

5.166.2 Constructor & Destructor Documentation

5.166.2.1 UniformSelection()

```
UniformSelection (
            const Population & population )   [inline]
```

Constructor.

Parameters

population	Population to select from

Definition at line 78 of file random-selection.hh.

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

- lib/hnco/algorithms/evolutionary-algorithms/random-selection.hh
- lib/hnco/algorithms/evolutionary-algorithms/random-selection.cc

5.167 UniversalFunction Class Reference

Universal function.

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

Public Member Functions

- virtual ∼**UniversalFunction** ()

    Destructor.
- virtual double **evaluate** (const bit_vector_t &boolean_vars, const std::vector< int > &integer_vars, const std::vector< double > &float_vars, const std::vector< std::complex< double > > &complex_vars, const std::vector< int > &categorical_vars, const std::vector< permutation_t > &permutation_vars)=0

    Evaluate the function.
- virtual void **display** (std::ostream &stream) const

    Display the function.
- virtual void **describe** (const bit_vector_t &boolean_vars, const std::vector< int > &integer_vars, const std::vector< double > &float_vars, const std::vector< std::complex< double > > &complex_vars, const std::vector< int > &categorical_vars, const std::vector< permutation_t > &permutation_vars, std::ostream &stream)

    Describe variables in the context of the function.

5.167.1 Detailed Description

Universal function.

A universal function is a function taking parameters of all types (boolean, integer, float, complex, categorical, permutation) and returning a double.

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

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

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

5.168 UniversalFunction Class Reference

Universal function.

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

Public Member Functions

- virtual ~**UniversalFunction** ()

  Destructor.
- virtual int **get_output_size** () const =0

  Get output size (number of objectives)
- virtual void **evaluate** (const bit_vector_t &boolean_vars, const std::vector< int > &integer_vars, const std::vector< double > &float_vars, const std::vector< std::complex< double > > &complex_vars, const std::vector< int > &categorical_vars, const std::vector< permutation_t > permutation_vars, value_t &value)=0

  Evaluate the function.
- virtual void **display** (std::ostream &stream) const

  Display the function.
- virtual void **describe** (const bit_vector_t &boolean_vars, const std::vector< int > &integer_vars, const std::vector< double > &float_vars, const std::vector< std::complex< double > > &complex_vars, const std::vector< int > &categorical_vars, const std::vector< permutation_t > permutation_vars, std::ostream &stream)

  Describe variables in the context of the function.

5.168.1 Detailed Description

Universal function.

A universal function is a function taking parameters of all types (boolean, integer, float, complex, categorical, permutation) and returning a double.

Definition at line 43 of file universal-function.hh.

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

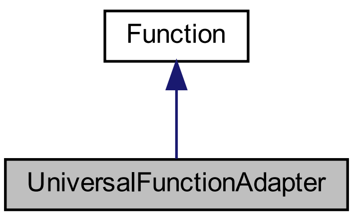
- lib/hnco/multiobjective/functions/universal-function.hh

5.169 UniversalFunctionAdapter Class Reference

Universal function adapter.

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

Inheritance diagram for UniversalFunctionAdapter:



Public Member Functions

- UniversalFunctionAdapter (UniversalFunction *fn, int num_boolean_vars, std::vector< representation::DyadicIntegerRepresent
int > > integer_reps, std::vector< representation::DyadicFloatRepresentation< double > > float_↩
reps, std::vector< representation::ComplexRepresentation< DoubleRep > > complex_reps, std::vector<
representation::LinearCategoricalRepresentation > categorical_reps, std::vector< representation::PermutationRepresentation
> permutation_reps)

  Constructor.
- int **get_bv_size** () const override

  Get bit vector size.
- double **evaluate** (const bit_vector_t &bv) override

  Evaluate a bit vector.
- void **display** (std::ostream &stream) const override

  Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override

  Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)

  Unpack bit vector into variables.

Private Attributes

- UniversalFunction * **_function**

  Universal function.
- std::vector< representation::DyadicIntegerRepresentation< int > > **_integer_reps**

  Integer representations.
- std::vector< representation::DyadicFloatRepresentation< double > > **_float_reps**

  Float representations.
- std::vector< representation::ComplexRepresentation< DoubleRep > > **_complex_reps**

  Complex representations.
- std::vector< representation::LinearCategoricalRepresentation > **_categorical_reps**

  Categorical representations.
- std::vector< representation::PermutationRepresentation > **_permutation_reps**

  Permuation representations.
- bit_vector_t **_boolean_vars**

  Boolean variables.
- std::vector< int > **_integer_vars**

  Integer variables.
- std::vector< double > **_float_vars**

  Float variables.
- std::vector< std::complex< double > > **_complex_vars**

  Complex variables.
- std::vector< int > **_categorical_vars**

  Categorical variables.
- std::vector< permutation_t > **_permutation_vars**

  Permutation variables.
- int **_bv_size**

  Bit vector size.

5.169.1 Detailed Description

Universal function adapter.

A universal function adapter turns a universal function into a regular hnco function defined on bit vectors.

Definition at line 45 of file universal-function-adapter.hh.

5.169.2 Constructor & Destructor Documentation

5.169.2.1 UniversalFunctionAdapter()

```
UniversalFunctionAdapter (
            UniversalFunction * fn,
            int num_boolean_vars,
            std::vector< representation::DyadicIntegerRepresentation< int > > integer_reps,
            std::vector< representation::DyadicFloatRepresentation< double > > float_reps,
            std::vector< representation::ComplexRepresentation< DoubleRep > > complex_reps,
            std::vector< representation::LinearCategoricalRepresentation > categorical_reps,
            std::vector< representation::PermutationRepresentation > permutation_reps ) [inline]
```

Constructor.

Parameters

fn	Universal function
num_boolean_vars	Number of boolean variables
integer_reps	Integer representations
float_reps	Float representations
complex_reps	Complex representations
categorical_reps	Categorical representations
permutation_reps	Permutation representations

Replace reps with {} if there is no corresponding variable. For example, if there is no categorical variable,

UniversalFunctionAdapter(fn, num_boolean_vars, integer_reps, float_reps, complex_reps, {}, permutation_reps)

Definition at line 134 of file universal-function-adapter.hh.

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

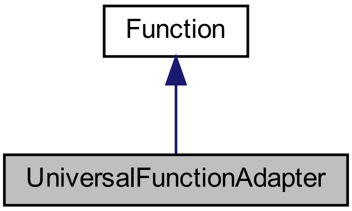
- lib/hnco/functions/universal-function-adapter.hh

5.170 UniversalFunctionAdapter Class Reference

Universal function adapter.

```
#include <hnco/multiobjective/functions/universal-function-adapter.hh>
```

Inheritance diagram for UniversalFunctionAdapter:



Public Member Functions

- UniversalFunctionAdapter (UniversalFunction ∗fn, int num_boolean_vars, std::vector< representation::DyadicIntegerRepresent
int > > integer_reps, std::vector< representation::DyadicFloatRepresentation< double > > float_↩
reps, std::vector< representation::ComplexRepresentation< DoubleRep > > complex_reps, std::vector< representation::LinearCategoricalRepresentation > categorical_reps, std::vector< representation::PermutationRepresentation > permutation_reps)

    Constructor.
- int **get_bv_size** () const override

    Get bit vector size.
- int **get_output_size** () const override

    Get output size (number of objectives)
- void **evaluate** (const bit_vector_t &bv, value_t &value) override

    Evaluate a bit vector.
- void **display** (std::ostream &stream) const override

    Display.
- void **describe** (const bit_vector_t &bv, std::ostream &stream) override

    Describe a bit vector.

Private Member Functions

- void **unpack** (const bit_vector_t &bv)

    Unpack bit vector into variables.

Private Attributes

- UniversalFunction ∗ **_function**

    Universal function.
- std::vector< representation::DyadicIntegerRepresentation< int > > **_integer_reps**

    Integer representations.
- std::vector< DoubleRep > **_float_reps**

    Float representations.

- std::vector< representation::ComplexRepresentation< DoubleRep > > **_complex_reps**

  Complex representations.
- std::vector< representation::LinearCategoricalRepresentation > **_categorical_reps**

  Categorical representations.
- std::vector< representation::PermutationRepresentation > **_permutation_reps**

  Permuation representations.
- bit_vector_t **_boolean_vars**

  Boolean variables.
- std::vector< int > **_integer_vars**

  Integer variables.
- std::vector< double > **_float_vars**

  Float variables.
- std::vector< std::complex< double > > **_complex_vars**

  Complex variables.
- std::vector< int > **_categorical_vars**

  Categorical variables.
- std::vector< permutation_t > **_permutation_vars**

  Permutation variables.
- int **_bv_size**

  Bit vector size.

5.170.1 Detailed Description

Universal function adapter.

A universal function adapter turns a universal function into a regular hnco function defined on bit vectors.

Definition at line 46 of file universal-function-adapter.hh.

5.170.2 Constructor & Destructor Documentation

5.170.2.1 UniversalFunctionAdapter()

```
UniversalFunctionAdapter (
            UniversalFunction * fn,
            int num_boolean_vars,
            std::vector< representation::DyadicIntegerRepresentation< int > > integer_reps,
            std::vector< representation::DyadicFloatRepresentation< double > > float_reps,
            std::vector< representation::ComplexRepresentation< DoubleRep > > complex_reps,
            std::vector< representation::LinearCategoricalRepresentation > categorical_reps,
            std::vector< representation::PermutationRepresentation > permutation_reps )  [inline]
```

Constructor.

Parameters

fn	Universal function
num_boolean_vars	Number of boolean variables
integer_reps	Integer representations
float_reps	Float representations
complex_reps	Complex representations
categorical_reps	Categorical representations

Replace reps with {} if there is no corresponding variable. For example, if there is no categorical variable,

UniversalFunctionAdapter(fn, num_boolean_vars, integer_reps, float_reps, complex_reps, {}, permutation_reps)

Definition at line 135 of file universal-function-adapter.hh.

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

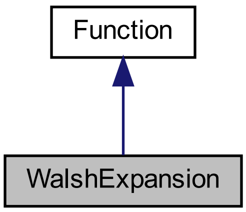
- lib/hnco/multiobjective/functions/universal-function-adapter.hh

5.171 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



Public Member Functions

- **WalshExpansion** ()
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- void **display** (std::ostream &stream) const override
  Display.
- void **set_terms** (const std::vector< function::WalshTerm > terms)
  Set terms.

Instance generators

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

Load and save instance

- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

Private Member Functions

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

  Save.

Private Attributes

- std::vector< function::WalshTerm > **_terms**

  Terms.

5.171.1 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 52 of file walsh-expansion.hh.

5.171.2 Member Function Documentation

5.171.2.1 generate()

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

Instance generator.

Parameters

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

Definition at line 85 of file walsh-expansion.hh.

5.171.2.2 load()

```
void load (
            std::string path ) [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

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

5.171.2.3 random()

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

Random instance.

The coefficients are sampled from the normal distribution.

Parameters

n	Size of bit vector
num_features	Number of feature vectors

Definition at line 111 of file walsh-expansion.hh.

5.171.2.4 save()

```
void save (
            std::string path ) const [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 137 of file walsh-expansion.hh.

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

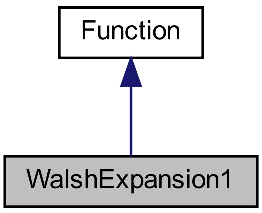
- lib/hnco/functions/collection/walsh/walsh-expansion.hh
- lib/hnco/functions/collection/walsh/walsh-expansion.cc

5.172 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

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

Inheritance diagram for WalshExpansion1:



Public Member Functions

- **WalshExpansion1** ()

  Constructor.

Instance generators

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

  Instance generator.
- void random (int n)

  Random instance.

Load and save instance

- void load (std::string path)

  Load instance.

- void save (std::string path) const
  Save instance.

Evaluation

- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.
- double **evaluate_incrementally** (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits) override
  Incrementally evaluate a bit vector.

Information about the function

- int **get_bv_size** () const override
  Get bit vector size.
- double **get_maximum** () const override
  Get the global maximum.
- bool has_known_maximum () const override
  Check for a known maximum.
- bool provides_incremental_evaluation () const override
  Check whether the function provides incremental evaluation.

Private Member Functions

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

Private Attributes

- std::vector< double > **_linear**
  Linear part.

5.172.1 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 49 of file walsh-expansion-1.hh.

5.172.2 Member Function Documentation

5.172.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

Definition at line 81 of file walsh-expansion-1.hh.

5.172.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 149 of file walsh-expansion-1.hh.

5.172.2.3 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 113 of file walsh-expansion-1.hh.

5.172.2.4 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 154 of file walsh-expansion-1.hh.

5.172.2.5 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

n	Size of bit vectors

Definition at line 95 of file walsh-expansion-1.hh.

5.172.2.6 save()

```
void save (
            std::string path ) const [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 120 of file walsh-expansion-1.hh.

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

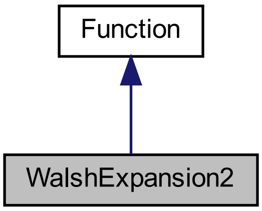
- lib/hnco/functions/collection/walsh/walsh-expansion-1.hh
- lib/hnco/functions/collection/walsh/walsh-expansion-1.cc

5.173 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

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

Inheritance diagram for WalshExpansion2:



Public Member Functions

- **WalshExpansion2** ()
  Constructor.
- int **get_bv_size** () const override
  Get bit vector size.
- double **evaluate** (const bit_vector_t &) override
  Evaluate a bit vector.

Instance generators

- template<class LinearGen , class QuadraticGen >
  void generate (int n, LinearGen linear_gen, QuadraticGen quadratic_gen)
  Instance generators.
- void random (int n)
  Instance generator.
- void generate_ising1_long_range (int n, double alpha)
  Generate one dimensional Ising model with long range interactions.
- void generate_ising1_long_range_periodic (int n, double alpha)
  Generate one dimensional Ising model with long range interactions and periodic boundary conditions.

Load and save instance

- void load (std::string path)
  Load instance.
- void save (std::string path) const
  Save instance.

Private Member Functions

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

  Serialize.
- void **resize** (int n)

  Resize data structures.

Private Attributes

- std::vector< double > **_linear**

  Linear part.
- std::vector< std::vector< double > > _quadratic

  Quadratic part.

5.173.1 Detailed Description

Walsh expansion of degree 2.

Its expression is of the form

$f(x) = \sum_i a_i(1 - 2x_i) + \sum_{i<j} a_{ij}(1 - 2x_i)(1 - 2x_j)$

or equivalently

$f(x) = \sum_i a_i(-1)^{x_i} + \sum_{i<j} a_{ij}(-1)^{x_i+x_j}$

Definition at line 49 of file walsh-expansion-2.hh.

5.173.2 Member Function Documentation

5.173.2.1 generate()

```
void generate (
            int n,
            LinearGen linear_gen,
            QuadraticGen quadratic_gen )  [inline]
```

Instance generators.

Parameters

n	Size of bit vectors
linear_gen	Generator for the linear part
quadratic_gen	Generator for the quadratic part

Definition at line 93 of file walsh-expansion-2.hh.

5.173.2.2 generate_ising1_long_range()

```
void generate_ising1_long_range (
            int n,
            double alpha )
```

Generate one dimensional Ising model with long range interactions.

Similar to a Dyson-Ising model except for the finite, instead of infinite, linear chain of spins.

Its expression is of the form

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

or equivalently

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

where $J(d_{ij})$ is the interaction between sites i and j, $d_{ij} = |i - j|$, and $J(n) = n^{-\alpha}$.

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

Parameters

n	Size of bit vectors
alpha	Exponential decay parameter

Definition at line 82 of file walsh-expansion-2.cc.

5.173.2.3 generate_ising1_long_range_periodic()

```
void generate_ising1_long_range_periodic (
            int n,
            double alpha )
```

Generate one dimensional Ising model with long range interactions and periodic boundary conditions.

Similar to a Dyson-Ising model except for the finite, instead of infinite, linear chain of spins.

Its expression is of the form

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

or equivalently

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

where $J(d_{ij})$ is the interaction between sites i and j, $d_{ij} = \min\{|i - j|, n - |i - j|\}$, and $J(n) = n^{-\alpha}$.

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

Parameters

n	Size of bit vectors
alpha	Exponential decay parameter

Definition at line 103 of file walsh-expansion-2.cc.

5.173.2.4 load()

```
void load (
            std::string path )  [inline]
```

Load instance.

Parameters

path	Path of the instance to load

Exceptions

std::runtime_error	

Definition at line 184 of file walsh-expansion-2.hh.

5.173.2.5 random()

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

Instance generator.

The weights are sampled from the normal distribution.

Parameters

n	Size of bit vector

Definition at line 115 of file walsh-expansion-2.hh.

5.173.2.6 save()

```
void save (
            std::string path ) const  [inline]
```

Save instance.

Parameters

path	Path of the instance to save

Exceptions

std::runtime_error	

Definition at line 191 of file walsh-expansion-2.hh.

5.173.3 Member Data Documentation

5.173.3.1 _quadratic

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

Quadratic part.

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

Definition at line 71 of file walsh-expansion-2.hh.

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

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

5.174 WalshTerm Struct Reference

Walsh transform term.

```
#include <hnco/functions/walsh-term.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.

5.174.1 Detailed Description

Walsh transform term.

Definition at line 35 of file walsh-term.hh.

5.174.2 Member Data Documentation

5.174.2.1 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 42 of file walsh-term.hh.

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

- lib/hnco/functions/walsh-term.hh

Index