HNCO

0.15

Generated by Doxygen 1.8.17

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

- algorithm

  Algorithms.
- exception

  Exceptions.
- function

  Functions defined on bit vectors.
- logging

  Logging.
- neighborhood

  Neighborhoods for local search.
- random

  Random numbers.

Classes

- class AffineMap

  Affine map.
- class HypercubeIterator

  Hypercube iterator.
- class Injection

  Injection.
- class Iterator

  Iterator over bit vectors
- class LinearMap

  Linear map.
- class Map

  Map

- class MapComposition

  Map composition.
- class Permutation

  Permutation.
- class Projection

  Projection.
- class StopWatch

  Stop watch.
- class Translation

  Translation.
- struct Transvection

  Transvection.
- class TsAffineMap

  Transvection sequence affine map.

Functions

- template<class 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)

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.

- typedef std::vector< bit_vector_t > bit_matrix_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_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 bit

- typedef char bit_t

    Bit.
- bit_t bit_flip (bit_t b)

    Flip bit.
- bit_t bit_random (double p)

    Sample a random bit.

Types and functions related to bit vectors

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.

- typedef std::vector< bit_t > bit_vector_t

    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_flip (bit_vector_t &x, int i)

    Flip a single bit.
- void bv_flip (bit_vector_t &x, const bit_vector_t &mask)

    Flip many bits.
- void bv_random (bit_vector_t &x)

    Sample a random bit vector.
- void bv_random (bit_vector_t &x, int k)

    Sample a random bit vector with given Hamming weight.
- void bv_add (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_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

- typedef std::vector< int > permutation_t

  Permutation type
- bool perm_is_valid (const permutation_t &permutation)

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

  Identity permutation.
- void perm_random (permutation_t &s)

  Sample a random permutation.

Types and functions related to sparse bit matrices

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

Input object parameters are passed by const reference.

- typedef std::vector< sparse_bit_vector_t > sparse_bit_matrix_t

  Sparse bit matrix.
- void sbm_display (const sparse_bit_matrix_t &sbm, std::ostream &stream)

  Display sparse bit matrix.
- void sbm_from_bm (sparse_bit_matrix_t &sbm, const bit_matrix_t &bm)

  Convert a bit matrix to a sparse bit matrix.
- void sbm_multiply (bit_vector_t &y, const sparse_bit_matrix_t &M, const bit_vector_t &x)

  Multiply a sparse bit matrix and a bit vector.

Types and functions related to sparse bit vectors

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

Input object parameters are passed by const reference.

- typedef std::vector< int > sparse_bit_vector_t

  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_flip (bit_vector_t &x, const sparse_bit_vector_t &sbv)

  Flip many bits of a bit vector.
- void sbv_display (const sparse_bit_vector_t &v, std::ostream &stream)

  Display sparse bit vector.
- sparse_bit_vector_t sbv_from_bv (const bit_vector_t &bv)

  Convert a bit vector to a sparse bit vector.

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.

- typedef std::vector< Transvection > transvection_sequence_t

  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.

4.1.1 Detailed Description

top-level HNCO namespace

4.1.2 Typedef Documentation

4.1.2.1 bit_t

```
typedef char bit_t
```

Bit.

A single bit is represented by a char.

Definition at line 48 of file bit-vector.hh.

4.1.2.2 sparse_bit_matrix_t

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

Sparse bit matrix.

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

Definition at line 48 of file sparse-bit-matrix.hh.

4.1.2.3 sparse_bit_vector_t

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

Sparse bit vector.

A sparse bit vector is represented as an 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 50 of file sparse-bit-vector.hh.

4.1.2.4 transvection_sequence_t

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

Transvection sequence.

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

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

Finite transvection sequences can then represent all invertible bit matrices.

Definition at line 165 of file transvection.hh.

4.1.3 Function Documentation

4.1.3.1 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.

4.1.3.2 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.

4.1.3.3 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.

4.1.3.4 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.

4.1.3.5 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 305 of file bit-matrix.cc.

4.1.3.6 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 349 of file bit-matrix.cc.

4.1.3.7 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 233 of file bit-matrix.cc.

4.1.3.8 bm_row_echelon_form()

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

Compute a row echelon form of a matrix.

Warning

A is modified by the function.

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

4.1.3.9 bm_solve()

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

Solve a linear system.

Solve the linear equation Ax = b.

Parameters

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 251 of file bit-matrix.cc.

4.1.3.10 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 284 of file bit-matrix.cc.

4.1.3.11 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.

4.1.3.12 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.

4.1.3.13 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 124 of file bit-vector.cc.

4.1.3.14 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.

4.1.3.15 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 201 of file bit-vector.cc.

4.1.3.16 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 232 of file bit-vector.cc.

4.1.3.17 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 216 of file bit-vector.cc.

4.1.3.18 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 156 of file bit-vector.cc.

4.1.3.19 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 169 of file bit-vector.cc.

4.1.3.20 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 184 of file bit-vector.cc.

4.1.3.21 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 143 of file bit-vector.cc.

4.1.3.22 is_in_range() [1/2]

```
bool hnco::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 45 of file util.hh.

4.1.3.23 is_in_range() [2/2]

```
bool hnco::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 56 of file util.hh.

4.1.3.24 perm_identity()

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

Identity permutation.

Warning

This function does not set the size of the permutation.

Definition at line 46 of file permutation.hh.

4.1.3.25 perm_random()

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

Sample a random permutation.

Warning

This function does not set the size of the permutation.

Definition at line 56 of file permutation.hh.

4.1.3.26 sbm_from_bm()

```
void sbm_from_bm (
            sparse_bit_matrix_t & sbm,
            const bit_matrix_t & bm )
```

Convert a bit matrix to a sparse bit matrix.

Parameters

sbm	Output sparse bit matrix
bm	Input bit matrix

Definition at line 40 of file sparse-bit-matrix.cc.

4.1.3.27 sbm_multiply()

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

Multiply a sparse bit matrix and a bit vector.

Computes y = Mx.

Parameters

y	Output bit vector
M	Input bit matrix
x	Input bit vector

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

4.1.3.28 sbv_flip()

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

Flip many bits of a bit vector.

Parameters

x	Input-output bit vector
sbv	Bits to flip

Definition at line 54 of file sparse-bit-vector.cc.

4.1.3.29 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 32 of file sparse-bit-vector.cc.

4.1.3.30 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 43 of file sparse-bit-vector.cc.

4.1.3.31 ts_is_valid() [1/2]

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

Check validity.

Parameters

ts	Transvection sequence

Definition at line 148 of file transvection.cc.

4.1.3.32 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 154 of file transvection.cc.

4.1.3.33 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 364 of file transvection.cc.

4.1.3.34 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 354 of file transvection.cc.

4.1.3.35 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 170 of file transvection.cc.

4.1.3.36 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 181 of file transvection.cc.

4.1.3.37 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 309 of file transvection.cc.

4.1.3.38 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 339 of file transvection.cc.

4.1.3.39 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 276 of file transvection.cc.

4.1.3.40 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 243 of file transvection.cc.

4.2 hnco::algorithm Namespace Reference

Algorithms.

Namespaces

- bm_pbil

    Boltzmann machine PBIL.
- fast_efficient_p3

    Algorithms from the FastEfficientP3 library.
- hea

    Herding evolutionary algorithm.

Classes

- class Algorithm

    Abstract search algorithm.
- class BiasedCrossover

    Biased crossover.
- class CompactGa

    Compact genetic algorithm.
- class CompleteSearch

    Complete search.
- class Crossover

Crossover
- class FirstAscentHillClimbing

  First ascent hill climbing.
- class GeneticAlgorithm

  Genetic algorithm.
- class Human

  Human.
- 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 Population

  Population
- class PvAlgorithm

  Probability vector algorithm.
- class RandomLocalSearch

  Random local search.
- class RandomSearch

  Random search.
- class RandomWalk

  Random walk.
- class Restart

  Restart.
- class SimulatedAnnealing

  Simulated annealing.
- class SteepestAscentHillClimbing

  Steepest ascent hill climbing.
- class TournamentSelection

  Population with tournament selection
- class Umda

  Univariate marginal distribution algorithm.
- class UniformCrossover

  Uniform crossover.

Typedefs

- typedef std::pair< bit_vector_t, double > solution_t

    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.

- typedef std::vector< double > pv_t

    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 hnco::algorithm::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 hnco::algorithm::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 hnco::algorithm::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::bm_pbil Namespace Reference

Boltzmann machine PBIL.

Classes

- class BmPbil

  Boltzmann machine PBIL.
- class Model

  Model of a Boltzmann machine
- class ModelParameters

  Parameters of a Boltzmann machine.

4.3.1 Detailed Description

Boltzmann machine PBIL.

4.4 hnco::algorithm::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.

4.4.1 Detailed Description

Algorithms from the FastEfficientP3 library.

4.5 hnco::algorithm::hea Namespace Reference

Herding evolutionary algorithm.

Classes

- class BitHerding

  Herding with bit features.
- struct BitMoment

  Moment for bit features.
- class Hea

  Herding evolutionary algorithm.
- class SpinHerding

  Herding with spin variables.
- struct SpinMoment

  Moment for spin variables.

4.5.1 Detailed Description

Herding evolutionary algorithm.

4.6 hnco::exception Namespace Reference

Exceptions.

Classes

- class Error

  Error
- class Exception

  Basic exception.
- class LastEvaluation

  Last evaluation.
- class LocalMaximumReached

  Local maximum reached.
- class MaximumReached

  Maximum reached.
- class SolutionFound

  Solution found.
- class TargetReached

  Target reached.

4.6.1 Detailed Description

Exceptions.

4.7 hnco::function Namespace Reference

Functions defined on bit vectors.

Namespaces

- controller
  Controllers.
- modifier
  Modifiers.
- representation
  Representations.

Classes

- class AbstractLabs
  Abstract class for low autocorrelation binary sequences.
- class AbstractMaxSat
  Abstract class for MaxSat-like functions.
- 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 LabsMeritFactor
  Low autocorrelation binary sequences merit factor.
- class LeadingOnes
  Leading ones.

- class LinearFunction
  Linear function.
- class LongPath
  Long path.
- class MaxNae3Sat
  Max not-all-equal 3SAT.
- class MaxSat
  MAX-SAT.
- class NearestNeighborIsingModel1
  Nearest neighbor Ising model in one dimension.
- class NearestNeighborIsingModel2
  Nearest neighbor Ising model in two dimensions.
- class Needle
  Needle in a haystack.
- class NkLandscape
  NK landscape.
- class OneMax
  OneMax.
- class Partition
  Partition.
- class Plateau
  Plateau.
- class Qubo
  Quadratic unconstrained binary optimization.
- class Ridge
  Ridge.
- class SinusSummationCancellation
  Summation cancellation with sinus.
- class SixPeaks
  Six Peaks.
- class SummationCancellation
  Summation cancellation.
- class Trap
  Trap.
- class WalshExpansion
  Walsh expansion.
- class WalshExpansion1
  Walsh expansion of degree 1.
- class WalshExpansion2
  Walsh expansion of degree 2.
- 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.
- 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.

4.7.1 Detailed Description

Functions defined on bit vectors.

4.7.2 Function Documentation

4.7.2.1 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.

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 31 of file function.cc.

4.8 hnco::function::controller Namespace Reference

Controllers.

Classes

- class Cache

  Cache.
- class CallCounter

Call counter.

- class Controller

  Function controller.
- class OnBudgetFunction

  CallCounter with a limited number of evaluations.
- class ProgressTracker

  ProgressTracker.
- class StopOnMaximum

  Stop on maximum.
- class StopOnTarget

  Stop on target.

Functions

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

  Insert formatted output.

4.8.1 Detailed Description

Controllers.

4.9 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.

4.9.1 Detailed Description

Modifiers.

4.10 hnco::function::representation Namespace Reference

Representations.

Classes

- class DyadicComplexRepresentation

    Dyadic complex representation.
- class DyadicIntegerRepresentation

    Dyadic integer representation.
- class DyadicRealRepresentation

    Dyadic real representation.
- class MultivariateFunction

    Multivariate function.
- class MultivariateFunctionAdapter

    Multivariate function adapter.
- class ParsedMultivariateFunction

    Parsed multivariate function.

Functions

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

    Check whether the difference is safe.

4.10.1 Detailed Description

Representations.

4.10.2 Function Documentation

4.10.2.1 difference_is_safe()

```
bool hnco::function::representation::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 185 of file representation.hh.

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::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.

4.12.1 Detailed Description

Neighborhoods for local search.

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

4.13 hnco::random Namespace Reference

Random numbers.

Classes

- struct Generator

    Random number generator.

4.13.1 Detailed Description

Random numbers.

Chapter 5

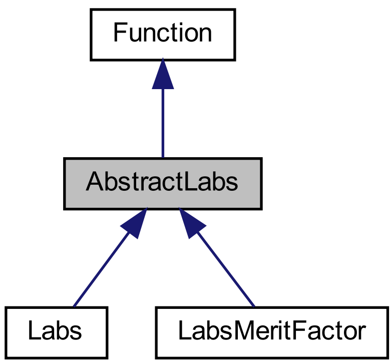
Class Documentation

5.1 AbstractLabs Class Reference

Abstract class for low autocorrelation binary sequences.

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

Inheritance diagram for AbstractLabs:



Public Member Functions

- AbstractLabs (int n)

    Constructor.
- int get_bv_size ()

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

    Compute autocorrelation.

Protected Attributes

- std::vector< int > _sequence

  Binary sequence written using 1 and -1.

5.1.1 Detailed Description

Abstract class for low autocorrelation binary sequences.

Definition at line 32 of file labs.hh.

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

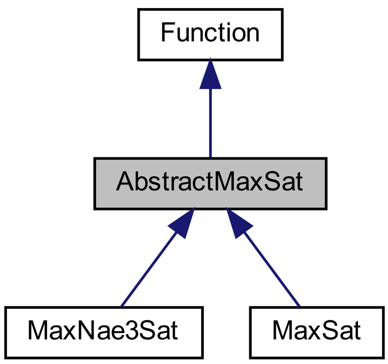
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.2 AbstractMaxSat Class Reference

Abstract class for MaxSat-like functions.

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

Inheritance diagram for AbstractMaxSat:



Public Member Functions

- AbstractMaxSat ()

  Default constructor.
- int get_bv_size ()

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

  Display the expression.
- virtual void load (std::istream &stream)

  Load an instance.
- virtual void save (std::ostream &stream)

  Save an instance.

Protected Attributes

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

    Expression.
- int _num_variables

    Number of variables.

5.2.1 Detailed Description

Abstract class for MaxSat-like functions.

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

5.2.2 Member Function Documentation

5.2.2.1 load()

```
void load (
            std::istream & stream )  [virtual]
```

Load an instance.

Exceptions

Error	

Reimplemented in MaxNae3Sat.

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

5.2.3 Member Data Documentation

5.2.3.1 _expression

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

Expression.

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

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

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

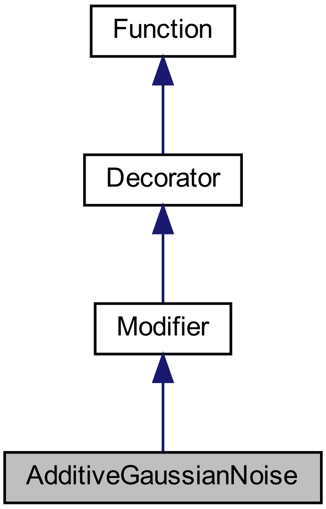
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.3 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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

Inheritance diagram for AdditiveGaussianNoise:



Public Member Functions

- AdditiveGaussianNoise (Function ∗function, double stddev)

  Constructor.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Information about the function

- int get_bv_size ()

  Get bit vector size.

Private Attributes

- std::normal_distribution< double > _dist

  Normal distribution.

Additional Inherited Members

5.3.1 Detailed Description

Additive Gaussian Noise.

Definition at line 170 of file modifier.hh.

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

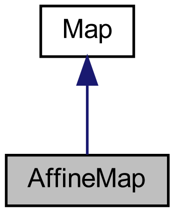
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.4 AffineMap Class Reference

Affine map.

`#include <hnco/map.hh>`

Inheritance diagram for AffineMap:



Public Member Functions

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

  Random instance.
- void map (const bit_vector_t &input, bit_vector_t &output)

  Map
- int get_input_size ()

  Get input size.
- int get_output_size ()

  Get output size.
- bool is_surjective ()

  Check for surjective map.

Private Member Functions

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

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

  Load.

Private Attributes

- bit_matrix_t _bm

  Bit matrix.
- bit_vector_t _bv

  Translation vector

Friends

- class **boost::serialization::access**

5.4.1 Detailed Description

Affine map.

An affine map f from 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 264 of file map.hh.

5.4.2 Member Function Documentation

5.4.2.1 is_surjective()

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

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 136 of file map.cc.

5.4.2.2 random()

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

Random instance.

Parameters

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

Exceptions

Error	

Definition at line 99 of file map.cc.

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

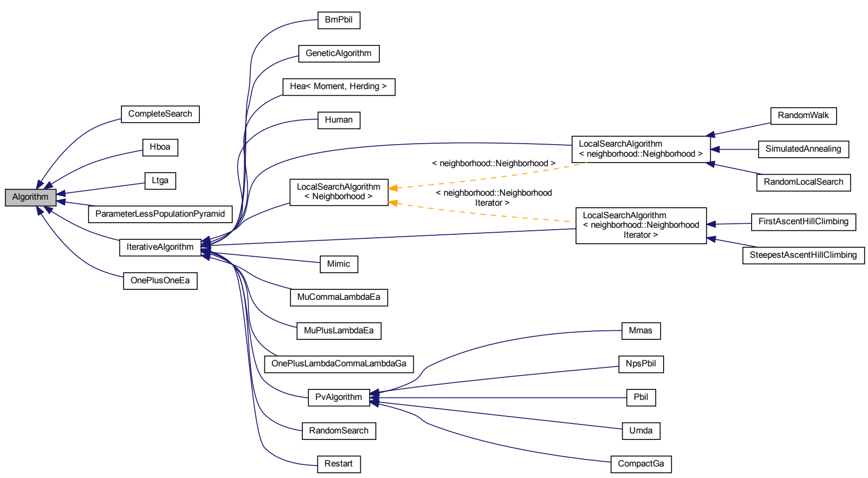
- lib/hnco/map.hh
- lib/hnco/map.cc

5.5 Algorithm Class Reference

Abstract search algorithm.

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

Inheritance diagram for Algorithm:



Public Member Functions

- Algorithm (int n)

    Constructor.
- virtual ∼Algorithm ()

    Destructor.

Optimization

- virtual void maximize (const std::vector< function::Function * > &functions)=0
  Maximize.
- virtual void finalize ()
  Finalize.

Getters

- int get_bv_size ()
  Get bit vector size.
- const solution_t & get_solution ()
  Get the solution.

Setters

- void set_log_context (logging::LogContext *log_context)
  Set the log context.

Protected Member Functions

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

Managing solution

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

Protected Attributes

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

Parameters

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

5.5.1 Detailed Description

Abstract search algorithm.

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

Definition at line 44 of file algorithm.hh.

5.5.2 Member Function Documentation

5.5.2.1 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 OnePlusOneEa, RandomLocalSearch, ParameterLessPopulationPyramid, Hboa, and Ltga.

Definition at line 141 of file algorithm.hh.

5.5.2.2 set_solution()

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

Set solution.

Warning

Evaluates the function once.

Definition at line 45 of file algorithm.cc.

5.5.2.3 update_solution()

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

Update solution (strict).

Warning

Evaluates the function once.

Definition at line 62 of file algorithm.cc.

5.5.3 Member Data Documentation

5.5.3.1 _functions

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

Functions.

Each thread has its own function.

Definition at line 55 of file algorithm.hh.

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

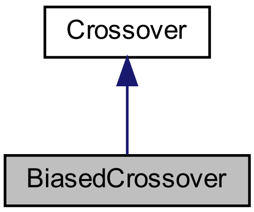
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

5.6 BiasedCrossover Class Reference

Biased crossover.

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

Inheritance diagram for BiasedCrossover:



Public Member Functions

- BiasedCrossover ()

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

    Breed.
- void set_bias (double b)

    Set bias.

Private Attributes

- std::bernoulli_distribution _bernoulli_dist

    Bernoulli distribution.

5.6.1 Detailed Description

Biased crossover.

Definition at line 75 of file crossover.hh.

5.6.2 Member Function Documentation

5.6.2.1 breed()

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

Breed.

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

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implements Crossover.

Definition at line 45 of file crossover.cc.

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

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

5.7 BitHerding Class Reference

Herding with bit features.

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

Public Types

- enum { DYNAMICS_MINIMIZE_NORM, DYNAMICS_MAXIMIZE_INNER_PRODUCT }

Public Member Functions

- BitHerding (int n)

  Constructor.
- void init ()

  Initialization.
- void sample (const BitMoment &target, bit_vector_t &x)

  Sample a bit vector.
- double error (const BitMoment &target)

  Compute the error.

Getters

- const BitMoment & get_delta ()

  Get delta.

Setters

- void set_randomize_bit_order (bool x)

  Randomize bit order.
- void set_dynamics (int x)

  Set the dynamics.
- void set_weight (double x)

  Set the weight of second order moments.

Protected Member Functions

- void compute_delta (const BitMoment &target)

  Compute delta.
- void sample_minimize_norm (const BitMoment &target, bit_vector_t &x)

  Sample a bit vector.
- void sample_maximize_inner_product (const BitMoment &target, bit_vector_t &x)

  Sample a bit vector.

Protected Attributes

- BitMoment _count
  Counter moment.
- BitMoment _delta
  Delta moment.
- permutation_t _permutation
  Permutation.
- std::uniform_int_distribution< int > _choose_bit
  Choose bit.
- int _time
  Time.

Parameters

- bool _randomize_bit_order = false
  Randomize bit order.
- int _dynamics = DYNAMICS_MINIMIZE_NORM
  Dynamics.
- double _weight = 1
  Weight of second order moments.

5.7.1 Detailed Description

Herding with bit features.

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

5.7.2 Member Enumeration Documentation

5.7.2.1 anonymous enum

`anonymous enum`

Enumerator

DYNAMICS_MINIMIZE_NORM	Dynamics defined as minimization of a norm.
DYNAMICS_MAXIMIZE_INNER_PRODUCT	Dynamics defined as maximization of an inner product.

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

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

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

5.8 BitMoment Struct Reference

Moment for bit features.

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

Public Member Functions

- BitMoment (int n)
  Constructor.
- void uniform ()
  Set the moment to that of the uniform distribution.
- void init ()
  Initialize.
- void add (const bit_vector_t &x)
  Accumulate a bit vector.
- void average (int count)
  Compute average.
- void update (const BitMoment &p, double rate)
  Update moment.
- void bound (double margin)
  Bound moment.
- double distance (const BitMoment &p) const
  Distance.
- double norm_2 () const
  Compute the norm 2.
- double diameter () const
  Compute the diameter.
- size_t size () const
  Size.
- void display (std::ostream &stream)
  Display.

Public Attributes

- std::vector< std::vector< double > > _moment
  Moment.
- double _weight = 1
  Weight of second order moments.

5.8.1 Detailed Description

Moment for bit features.

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

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

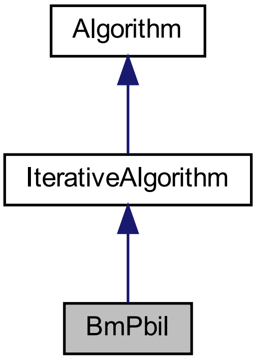
- lib/hnco/algorithms/hea/bit-moment.hh
- lib/hnco/algorithms/hea/bit-moment.cc

5.9 BmPbil Class Reference

Boltzmann machine PBIL.

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

Inheritance diagram for BmPbil:



Public Types

- enum { SAMPLING_ASYNCHRONOUS, SAMPLING_ASYNCHRONOUS_FULL_SCAN, SAMPLING_SYNCHRONOUS }
- enum { RESET_NO_RESET, RESET_ITERATION, RESET_BIT_VECTOR }

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 (int x)
  Set the sampling mode.
- void set_mc_reset_strategy (int x)
  Set the MC reset strategy.

Setters for logging

- void set_log_norm_infinite (bool x)
  Log infinite norm of the model parameters.
- void set_log_norm_l1 (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.
- Model _model

    Model.
- ModelParameters _parameters_all

    Parameters averaged over all individuals.
- ModelParameters _parameters_best

    Parameters averaged over selected individuals.
- ModelParameters _parameters_worst

    Parameters averaged over negatively selected individuals.
- std::uniform_int_distribution< 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.
- int _sampling = SAMPLING_ASYNCHRONOUS

Sampling mode.
- int _mc_reset_strategy = RESET_NO_RESET

  MC reset strategy.

Logging

- bool _log_norm_infinite = false

  Log infinite norm of the model parameters.
- bool _log_norm_l1 = false

  Log 1-norm of the model parameters.

5.9.1 Detailed Description

Boltzmann machine PBIL.

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

Reference:

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

Definition at line 49 of file bm-pbil.hh.

5.9.2 Member Enumeration Documentation

5.9.2.1 anonymous enum

```
anonymous enum
```

Enumerator

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

Definition at line 54 of file bm-pbil.hh.

5.9.2.2 anonymous enum

```
anonymous enum
```

Enumerator

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

Definition at line 82 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 216 of file bm-pbil.hh.

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

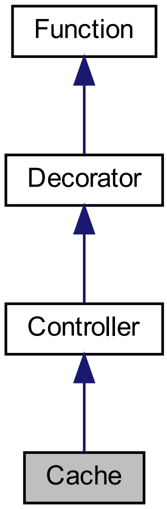
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

5.10 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 ()

    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 364 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 383 of file controller.hh.

5.10.3 Member Function Documentation

5.10.3.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Controller.

Definition at line 392 of file controller.hh.

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

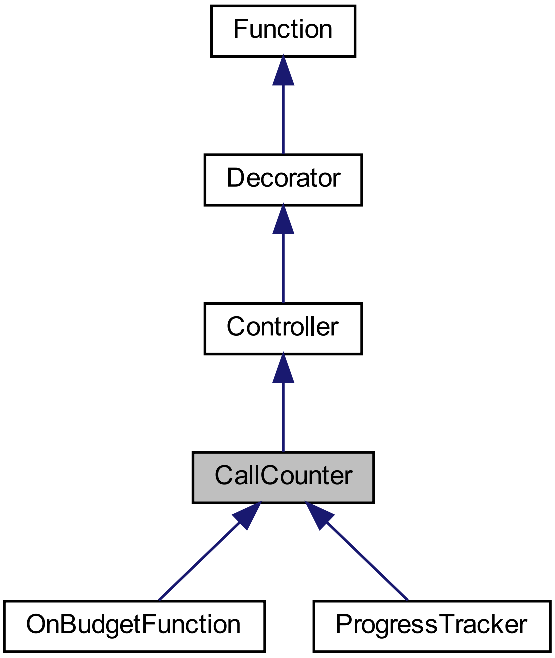
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

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 &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &x, 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 175 of file controller.hh.

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

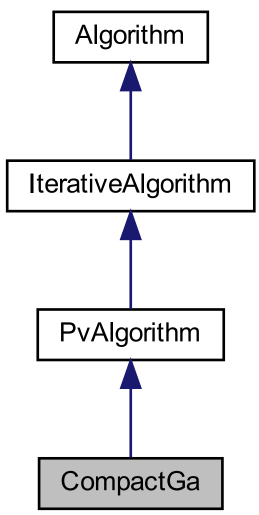
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.12 CompactGa Class Reference

Compact genetic algorithm.

```
#include <hnco/algorithms/pv/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.

5.12.1 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:

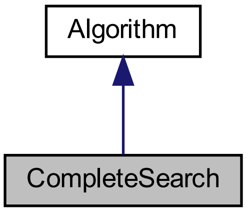
- lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

5.13 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

5.13.1 Detailed Description

Complete search.

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

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

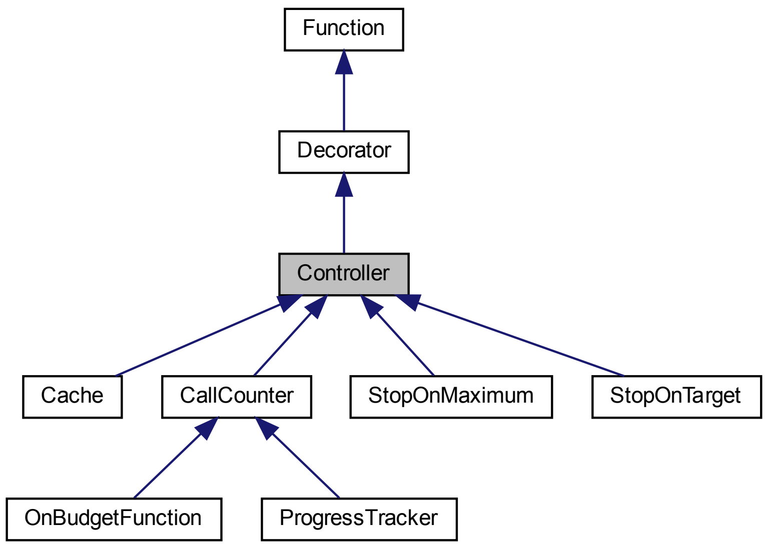
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

5.14 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 ()

  Get bit vector size.
- double get_maximum ()

  Get the global maximum.
- bool has_known_maximum ()

  Check for a known maximum.
- bool provides_incremental_evaluation ()

  Check whether the function provides incremental evaluation.

Evaluation

- double evaluate_safely (const bit_vector_t &x)

  Safely evaluate a bit vector.

Additional Inherited Members

5.14.1 Detailed Description

Function controller.

Definition at line 41 of file controller.hh.

5.14.2 Member Function Documentation

5.14.2.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true if the decorated function does

Reimplemented from Function.

Reimplemented in Cache.

Definition at line 66 of file controller.hh.

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

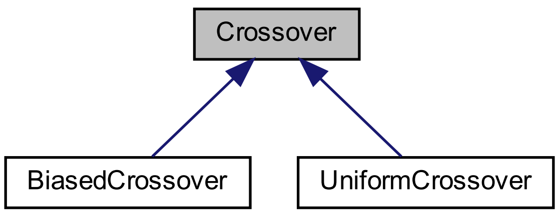
- lib/hnco/functions/controllers/controller.hh

5.15 Crossover Class Reference

Crossover

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

Inheritance diagram for Crossover:



Public Member Functions

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

5.15.1 Detailed Description

Crossover

Definition at line 35 of file crossover.hh.

5.15.2 Member Function Documentation

5.15.2.1 breed()

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

Breed.

The offspring is the crossover of two parents.

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implemented in BiasedCrossover, and UniformCrossover.

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

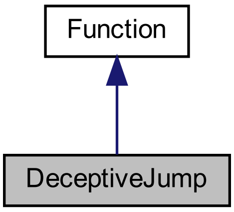
- lib/hnco/algorithms/ea/crossover.hh

5.16 DeceptiveJump Class Reference

Deceptive jump.

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

Inheritance diagram for DeceptiveJump:



Public Member Functions

- DeceptiveJump (int bv_size, int gap)

    Constructor.
- int get_bv_size ()

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

    Evaluate a bit vector.
- bool has_known_maximum ()

    Check for a known maximum.
- double get_maximum ()

    Get the global maximum.

Private Attributes

- int _bv_size

    Bit vector size.
- int _gap

    Gap.

5.16.1 Detailed Description

Deceptive jump.

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

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 85 of file jump.hh.

5.16.2 Member Function Documentation

5.16.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size + _gap

Reimplemented from Function.

Definition at line 111 of file jump.hh.

5.16.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 107 of file jump.hh.

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

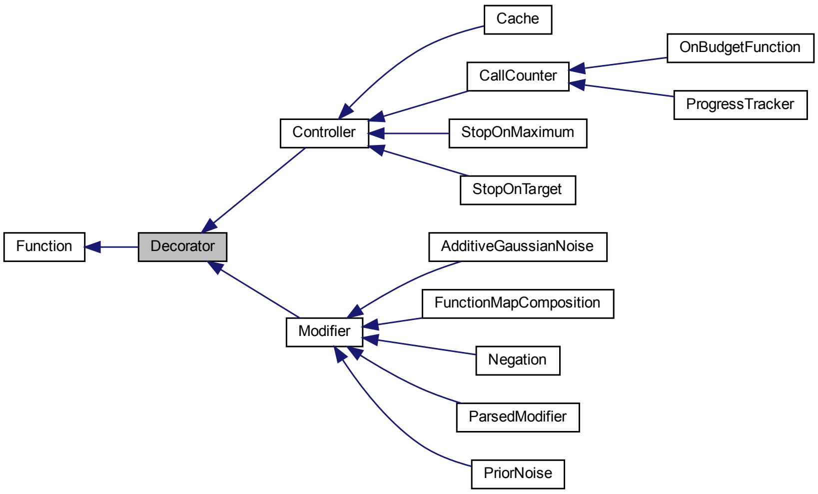
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

5.17 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)
  Display.
- void describe (const bit_vector_t &x, std::ostream &stream)
  Describe a bit vector.

Protected Attributes

- Function ∗ _function
  Decorated function.

5.17.1 Detailed Description

Function decorator

Definition at line 33 of file decorator.hh.

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

- lib/hnco/functions/decorator.hh

5.18 DyadicComplexRepresentation< T > Class Template Reference

Dyadic complex representation.

`#include <hnco/functions/representations/representation.hh>`

Public Types

- typedef std::complex< T > value_type
  Type of represented value.

Public Member Functions

- DyadicComplexRepresentation (int num_bits, T lower_bound, T upper_bound)
  Constructor.
- int size ()
  Size of the representation.
- value_type unpack (const bit_vector_t &bv, int start)
  Unpack bit vector into a value.
- void display (std::ostream &stream)
  Display.

Static Public Member Functions

- static double to_double (value_type z)
  Cast a complex value to a double.

Private Attributes

- DyadicRealRepresentation< T > _real_part
  Representation of the real part.
- DyadicRealRepresentation< T > _imaginary_part
  Representation of the imaginary part.

5.18.1 Detailed Description

template<class T>
class hnco::function::representation::DyadicComplexRepresentation< T >

Dyadic complex representation.

Definition at line 117 of file representation.hh.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 DyadicComplexRepresentation()

```
DyadicComplexRepresentation (
            int num_bits,
            T lower_bound,
            T upper_bound )  [inline]
```

Constructor.

Parameters

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

Definition at line 143 of file representation.hh.

5.18.3 Member Function Documentation

5.18.3.1 to_double()

```
static double to_double (
            value_type z )  [inline], [static]
```

Cast a complex value to a double.

Parameters

z	Complex number

Returns

the squared magnitude of z

Definition at line 135 of file representation.hh.

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

- lib/hnco/functions/representations/representation.hh

5.19 DyadicIntegerRepresentation$<$ T $>$ Class Template Reference

Dyadic integer representation.

```
#include <hnco/functions/representations/representation.hh>
```

Public Types

- typedef T value_type

  Type of represented value.

Public Member Functions

- DyadicIntegerRepresentation (int num_bits, T lower_bound, T upper_bound)
    Constructor.
- DyadicIntegerRepresentation (T lower_bound, T upper_bound)
    Constructor.
- int size ()
    Size of the representation.
- value_type unpack (const bit_vector_t &bv, int start)
    Unpack bit vector into a value.
- void display (std::ostream &stream)
    Display.

Static Public Member Functions

- static double to_double (T x)
    Cast a T value to a double.

Private Member Functions

- void set_num_bits_complete (T lower_bound, T upper_bound)
    The the number of bits of a complete representation.

Private Attributes

- int _num_bits
    Number of bits.
- int _num_bits_complete
    Number of bits for a complete representation.
- T _lower_bound
    Lower bound of the search interval.
- T _upper_bound
    Upper bound of the search interval.

5.19.1 Detailed Description

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

Dyadic integer representation.

Definition at line 207 of file representation.hh.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 DyadicIntegerRepresentation() [1/2]

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

Constructor.

Parameters

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

Definition at line 245 of file representation.hh.

5.19.2.2 DyadicIntegerRepresentation() [2/2]

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

Constructor.

Parameters

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

Definition at line 263 of file representation.hh.

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

- lib/hnco/functions/representations/representation.hh

5.20 DyadicRealRepresentation< T > Class Template Reference

Dyadic real representation.

`#include <hnco/functions/representations/representation.hh>`

Public Types

- typedef T value_type
  Type of represented value.

Public Member Functions

- DyadicRealRepresentation (int num_bits, T lower_bound, T upper_bound)
  Constructor.
- int size ()
  Size of the representation.
- value_type unpack (const bit_vector_t &bv, int start)
  Unpack bit vector into a value.
- void display (std::ostream &stream)
  Display.

Static Public Member Functions

- static double to_double (T x)

  Cast a T value to a double.

Private Member Functions

- T affine_transformation (T x)

  Affine transformation.

Private Attributes

- std::vector< T > _lengths

  Lengths of dyadic intervals.
- T _lower_bound

  Lower bound of the search interval.
- T _length

  Length of the search interval.

5.20.1 Detailed Description

template<class T>
class hnco::function::representation::DyadicRealRepresentation< T >

Dyadic real representation.

Definition at line 42 of file representation.hh.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 DyadicRealRepresentation()

```
DyadicRealRepresentation (
            int num_bits,
            T lower_bound,
            T upper_bound )  [inline]
```

Constructor.

Parameters

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

Definition at line 70 of file representation.hh.

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

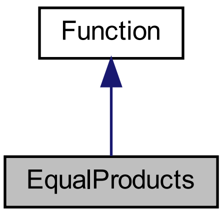
- lib/hnco/functions/representations/representation.hh

5.21 EqualProducts Class Reference

Equal products.

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

Inheritance diagram for EqualProducts:



Public Member Functions

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

Instance generators

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

Private Member Functions

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

Private Attributes

- std::vector< double > _numbers

    Numbers.

Friends

- class **boost::serialization::access**

5.21.1 Detailed Description

Equal products.

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

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

Reference:

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

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

5.21.2 Member Function Documentation

5.21.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Number generator

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

5.21.2.2 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 109 of file equal-products.hh.

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

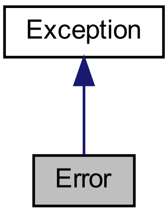
- lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

5.22 Error Class Reference

Error

`#include <hnco/exception.hh>`

Inheritance diagram for Error:



Public Member Functions

- Error ()
  Constructor.
- Error (const std::string &s)
  Constructor.
- virtual ∼Error ()
  Destructor.
- virtual const char ∗ what () const
  Get message.

Protected Attributes

- std::string _what

  Message.

5.22.1 Detailed Description

Error

Definition at line 97 of file exception.hh.

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

- lib/hnco/exception.hh

5.23 ProgressTracker::Event Struct Reference

Event

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

Public Attributes

- int num_evaluations

  Number of evaluations.
- double value

  Value.

5.23.1 Detailed Description

Event

Definition at line 224 of file controller.hh.

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

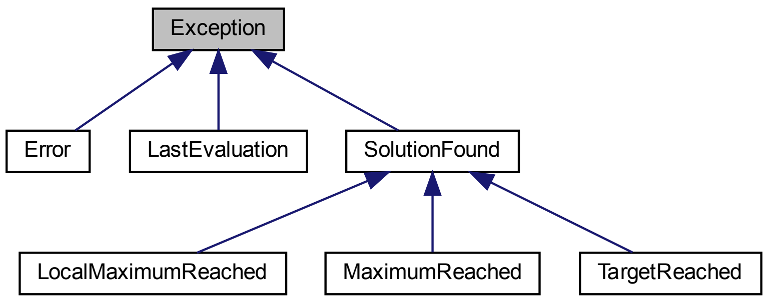
- lib/hnco/functions/controllers/controller.hh

5.24 Exception Class Reference

Basic exception.

`#include <hnco/exception.hh>`

Inheritance diagram for Exception:



5.24.1 Detailed Description

Basic exception.

Definition at line 37 of file exception.hh.

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

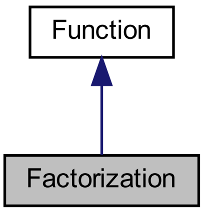
- lib/hnco/exception.hh

5.25 Factorization Class Reference

Factorization.

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

Inheritance diagram for Factorization:



Public Member Functions

- Factorization ()

  Constructor.
- Factorization (const std::string number)

  Constructor.
- ∼Factorization ()

  Destructor.
- void load (std::istream &stream)

  Load an instance.
- int get_bv_size ()

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

  Evaluate a bit vector.
- void display (std::ostream &stream)

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

  Describe a bit vector.

Private Member Functions

- void init ()

  Init GMP data structures.
- void clear ()

  Clear GMP data structures.
- void set_number (const std::string number)

  Set number.
- void convert (const bit_vector_t &x)

  Convert a bit vector into two numbers.

Private Attributes

- mpz_t _number

  Number to factorize.
- mpz_t _first_factor

  First factor.
- mpz_t _second_factor

  Second factor.
- mpz_t _product

  Product.
- std::string _first_factor_string

  First factor in binary form.
- std::string _second_factor_string

  Secon factor in binary form.
- size_t _number_size

  Number size in bits.
- size_t _first_factor_size

  First factor size in bits.
- size_t _second_factor_size

  Second factor size in bits.
- int _bv_size

  Bit vector size.

5.25.1 Detailed Description

Factorization.

Reference:

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

http://gmplib.org/.

Definition at line 28 of file factorization.hh.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 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.

5.25.3 Member Function Documentation

5.25.3.1 load()

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

Load an instance.

Warning

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

Exceptions

Error	

Definition at line 37 of file factorization.cc.

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

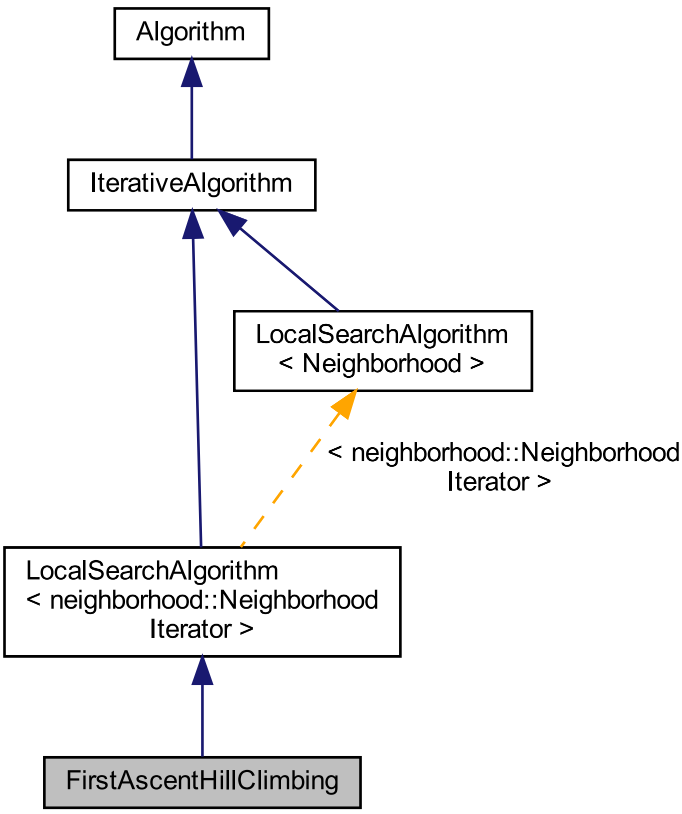
- lib/hnco/functions/factorization.hh
- lib/hnco/functions/factorization.cc

5.26 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

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

Inheritance diagram for FirstAscentHillClimbing:



Public Member Functions

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

    Constructor.

Protected Member Functions

- void iterate () override

    Single iteration.

Additional Inherited Members

5.26.1 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:

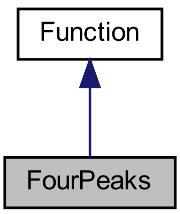
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc

5.27 FourPeaks Class Reference

Four Peaks.

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

Inheritance diagram for FourPeaks:



Public Member Functions

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

Private Attributes

- int _bv_size
  Bit vector size.
- int _threshold
  Threshold.
- int _maximum
  Maximum.

5.27.1 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.

5.27.2 Member Function Documentation

5.27.2.1 get_maximum()

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

Get the global maximum.

Returns

2 ∗ _bv_size - _threshold - 1

Reimplemented from Function.

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

5.27.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

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

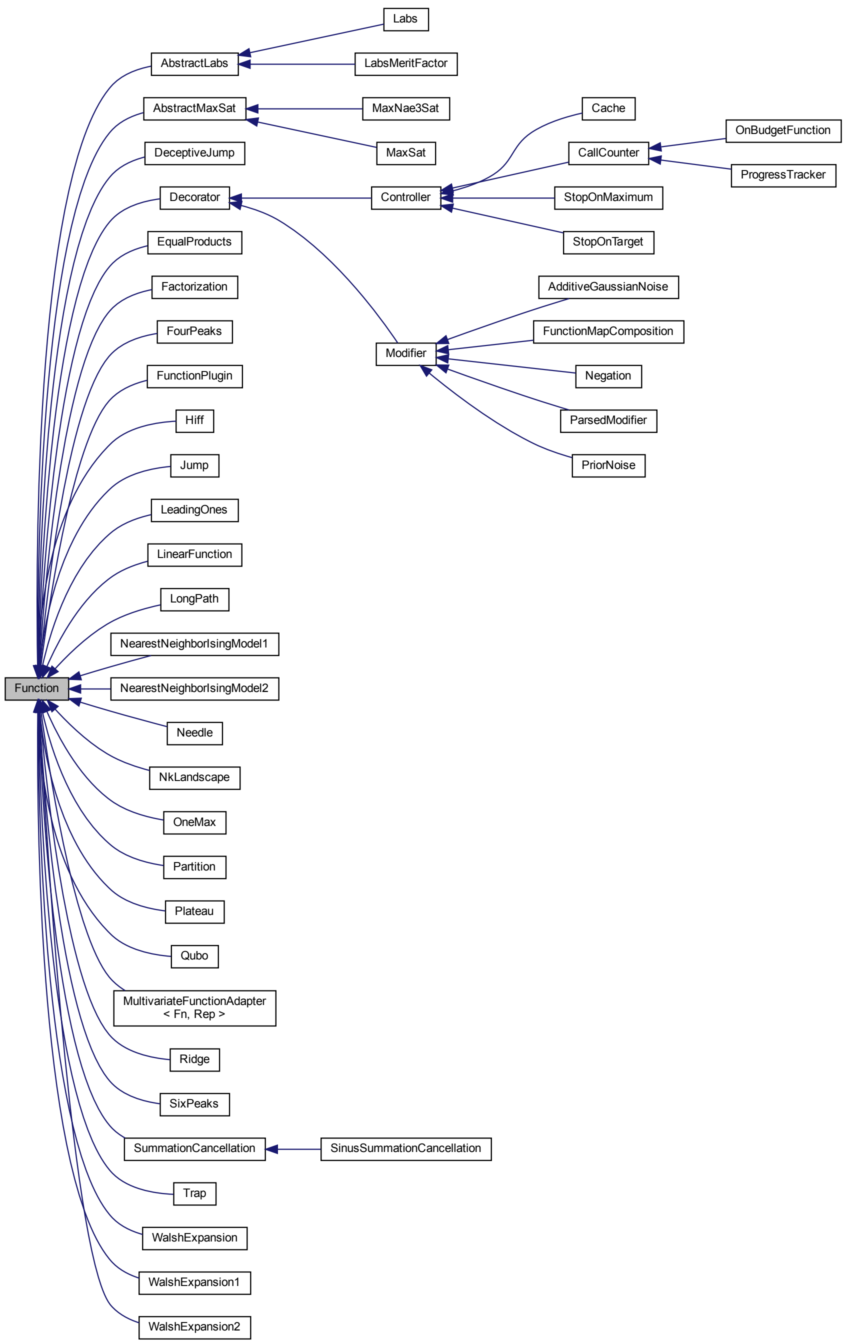
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

5.28 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 ()=0
  Get bit vector size.
- virtual double get_maximum ()
  Get the global maximum.
- virtual bool has_known_maximum ()
  Check for a known maximum.
- virtual bool provides_incremental_evaluation ()
  Check whether the function provides incremental evaluation.

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

5.28.1 Detailed Description

Function

Definition at line 45 of file function.hh.

5.28.2 Member Function Documentation

5.28.2.1 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 SinusSummationCancellation, SummationCancellation, MultivariateFunctionAdapter< Fn, Rep >, Cache, OnBudgetFunction, ProgressTracker, Plateau, Ridge, CallCounter, AdditiveGaussianNoise, Hiff, WalshExpansion2, NearestNeighborIsingModel2, StopOnTarget, NearestNeighborIsingModel1, FunctionMapComposition, Needle, SixPeaks, MaxNae3Sat, WalshExpansion, NkLandscape, EqualProducts, Partition, LeadingOnes, StopOnMaximum, MaxSat, WalshExpansion1, Factorization, DeceptiveJump, LabsMeritFactor, LinearFunction, Qubo, Negation, PriorNoise, FourPeaks, OneMax, Trap, LongPath, Labs, ParsedModifier, FunctionPlugin, and Jump.

5.28.2.2 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

Error	

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

Definition at line 95 of file function.hh.

5.28.2.3 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 109 of file function.hh.

5.28.2.4 get_maximum()

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

Get the global maximum.

Exceptions

Error	

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

Definition at line 61 of file function.hh.

5.28.2.5 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

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

Definition at line 71 of file function.hh.

5.28.2.6 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 OnBudgetFunction, ProgressTracker, CallCounter, StopOnTarget, and StopOnMaximum.

Definition at line 115 of file function.hh.

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

- lib/hnco/functions/function.hh

5.29 FunctionFactory Class Reference

Function factory.

```
#include </home/arnaud/projets/hnco/src/hnco/app/make-function.hh>
```

Public Member Functions

- hnco::function::Function * make_function (Options &options)

  Make a function.
- hnco::function::Function * make_function_controller (hnco::function::Function *function, const Options &options)

  Make a function controller.
- hnco::Map * get_map ()

  Get map.
- hnco::function::controller::Cache * get_cache ()

  Get cache.
- hnco::function::controller::ProgressTracker * get_tracker ()

  Get tracker.

Private Member Functions

- hnco::function::Function * make_function_modifier (hnco::function::Function *function, Options &options)

  Make a function modifier.

Private Attributes

- hnco::Map * _map = 0

  Map.
- hnco::function::controller::Cache * _cache = 0

  Cache.
- hnco::function::controller::ProgressTracker * _tracker = 0

  Tracker.

5.29.1 Detailed Description

Function factory.

Definition at line 30 of file make-function.hh.

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

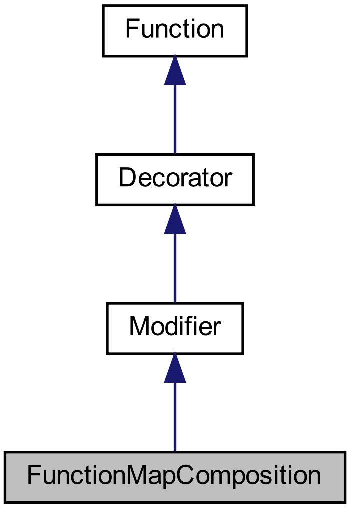
- app/make-function.hh
- app/make-function.cc

5.30 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, Map ∗map)

  Constructor.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Information about the function

- int get_bv_size ()

  Get bit vector size.
- double get_maximum ()

  Get the global maximum.
- bool has_known_maximum ()

  Check for a known maximum.

Display

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

  Describe a bit vector.

Private Attributes

- Map * _map
  Map.
- bit_vector_t _bv
  Image of bit vectors under the map.

Additional Inherited Members

5.30.1 Detailed Description

Composition of a function and a map.

Definition at line 100 of file modifier.hh.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 FunctionMapComposition()

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

Constructor.

Precondition

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

Exceptions

Error	

Definition at line 115 of file modifier.hh.

5.30.3 Member Function Documentation

5.30.3.1 get_maximum()

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

Get the global maximum.

Exceptions

Error	

Reimplemented from Function.

Definition at line 135 of file modifier.hh.

5.30.3.2 has_known_maximum()

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

Check for a known maximum.

Returns

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

Reimplemented from Function.

Definition at line 145 of file modifier.hh.

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

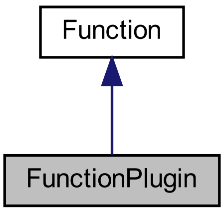
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.31 FunctionPlugin Class Reference

Function plugin

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

Inheritance diagram for FunctionPlugin:



Public Member Functions

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

    Constructor.
- ∼FunctionPlugin ()

    Destructor.
- int get_bv_size ()

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

    Evaluate a bit vector.

Private Types

- typedef double(∗ extern_function_t) (const bit_t ∗, size_t)

    Type of an extern function.

Private Attributes

- int _bv_size

    Bit vector size.
- void ∗ _handle

    Handle returned by dlopen.
- extern_function_t _extern_function

    Extern function.

5.31.1 Detailed Description

Function plugin

Definition at line 34 of file plugin.hh.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 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 33 of file plugin.cc.

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

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

5.32 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.

5.32.1 Detailed Description

Random number generator.

Definition at line 34 of file random.hh.

5.32.2 Member Function Documentation

5.32.2.1 reset()

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

Reset engine.

Using static member seed.

Definition at line 45 of file random.cc.

5.32.2.2 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:

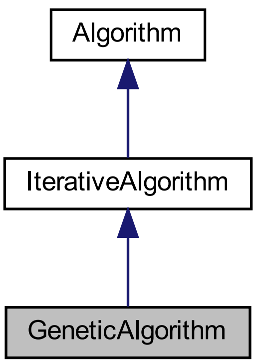
- lib/hnco/random.hh
- lib/hnco/random.cc

5.33 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



Public Member Functions

- GeneticAlgorithm (int n, int mu)

  Constructor.

Setters

- void set_mutation_rate (double p)

  Set the mutation rate.
- void set_crossover_probability (double x)

  Set the crossover probability.
- void set_tournament_size (int x)

  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

- TournamentSelection _parents

  Parents.
- TournamentSelection _offsprings

  Offsprings.
- 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.

5.33.1 Detailed Description

Genetic algorithm.

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

Reference:

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

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

5.33.2 Constructor & Destructor Documentation

5.33.2.1 GeneticAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors
mu	Population size

Definition at line 108 of file genetic-algorithm.hh.

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

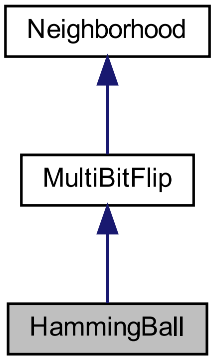
- lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

5.34 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

5.34.1 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 300 of file neighborhood.hh.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 HammingBall()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the ball

Definition at line 316 of file neighborhood.hh.

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

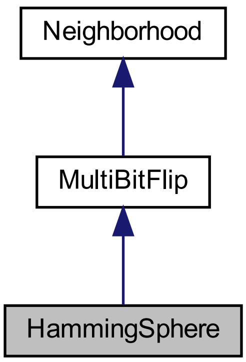
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.35 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

5.35.1 Detailed Description

Hamming sphere.

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

Definition at line 333 of file neighborhood.hh.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 HammingSphere()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of the sphere

Definition at line 349 of file neighborhood.hh.

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

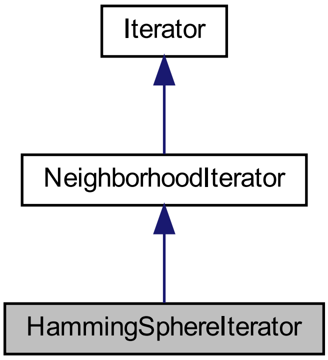
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.36 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 ()
  Has next bit vector.
- const bit_vector_t & next ()
  Next bit vector.

Private Attributes

- bit_vector_t _mask
  Mutation mask.
- int _radius
  Radius of the ball.
- int _index
  Index of the next bit to shift to the right.
- int _weight
  Partial Hamming weight.

Additional Inherited Members

5.36.1 Detailed Description

Hamming sphere neighborhood iterator.

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

0 ... 0 1 ... 1 0 ...

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

1 ... 1 0 ... 0 1 ...

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

5.36.2 Constructor & Destructor Documentation

5.36.2.1 HammingSphereIterator()

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

Constructor.

Parameters

n	Size of bit vectors
r	Radius of Hamming Ball

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

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

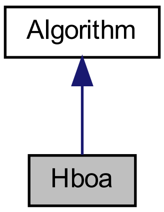
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.37 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

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

Inheritance diagram for Hboa:



Public Member Functions

- Hboa (int n)

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

    Maximize.
- void finalize ()

    Finalize.
- void set_population_size (int n)

    Set population size.

Private Attributes

- std::unique_ptr< Implementation > _pimpl

    Pointer to implementation.
- int _population_size = 10

    Population size.

Additional Inherited Members

5.37.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 48 of file hboa.hh.

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

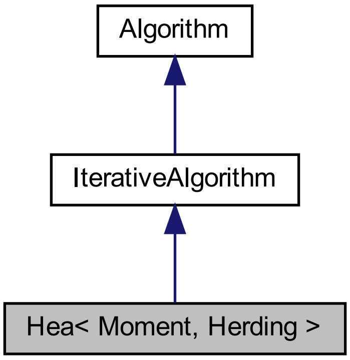
- lib/hnco/algorithms/fast-efficient-p3/hboa.hh
- lib/hnco/algorithms/fast-efficient-p3/hboa.cc

5.38 Hea< Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



Public Types

- enum {
  LOG_ERROR, LOG_DTU, LOG_DELTA, LOG_SELECTION,
  LOG_MOMENT_MATRIX, **LAST_LOG** }
- typedef std::bitset< LAST_LOG > log_flags_t

  Type for log flags.

Public Member Functions

- Hea (int n, int population_size)

  Constructor.

Setters

- void set_herding (Herding ∗x)

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

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

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

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

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

  Set the bound moment after update.
- void set_weight (double weight)

  Set weight.
- void set_log_flags (const log_flags_t &lf)

  Set log flags.

Protected Member Functions

Loop

- void init () override
  Initialization.
- void iterate () override
  Single iteration.
- void log () override
  Log.

Protected Attributes

- Moment _target
  Moment.
- Moment _selection
  Moment of selected individuals.
- Moment _uniform
  Uniform moment.
- algorithm::Population _population
  Population.
- Herding * _herding
  Herding.

Logging

- double _error_cache
  Error cache.
- double _dtu_cache
  Distance to uniform cache.
- double _delta_cache
  Delta cache.
- double _selection_cache
  Selection distance cache.
- log_flags_t _log_flags
  Log flags.

Parameters

- double _margin
  Moment margin.
- int _selection_size = 1
  Selection size.
- int _reset_period = 0
  Reset period.
- double _learning_rate = 1e-4
  Learning rate.
- bool _bound_moment = false
  Bound moment after update.

5.38.1 Detailed Description

template<class Moment, class Herding>
class hnco::algorithm::hea::Hea< Moment, 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 52 of file hea.hh.

5.38.2 Member Enumeration Documentation

5.38.2.1 anonymous enum

`anonymous enum`

Enumerator

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

Definition at line 56 of file hea.hh.

5.38.3 Constructor & Destructor Documentation

5.38.3.1 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 229 of file hea.hh.

5.38.4 Member Function Documentation

5.38.4.1 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 266 of file hea.hh.

5.38.4.2 set_selection_size()

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

Set the selection size.

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

Definition at line 258 of file hea.hh.

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

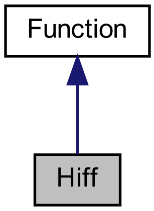
- lib/hnco/algorithms/hea/hea.hh

5.39 Hiff Class Reference

Hierarchical if and only if.

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

Inheritance diagram for Hiff:



Public Member Functions

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

Private Attributes

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

5.39.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 170 of file theory.hh.

5.39.2 Member Function Documentation

5.39.2.1 get_maximum()

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

Get the global maximum.

Returns

(i + 1) ∗ 2^i where 2^i = _bv_size

Reimplemented from Function.

Definition at line 196 of file theory.hh.

5.39.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 192 of file theory.hh.

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

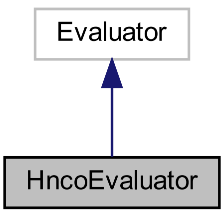
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.40 HncoEvaluator Class Reference

Evaluator for HNCO functions.

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

Inheritance diagram for HncoEvaluator:



Public Member Functions

- HncoEvaluator (hnco::function::Function ∗function)

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

    Evaluate a bit vector.

Private Attributes

- hnco::function::Function ∗ _function

    HNCO function.
- hnco::bit_vector_t _bv

    Argument of HNCO function.

5.40.1 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:

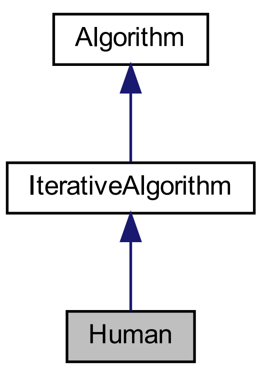
- lib/hnco/algorithms/fast-efficient-p3/hnco-evaluator.hh

5.41 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.

5.41.1 Detailed Description

Human.

Definition at line 31 of file human.hh.

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

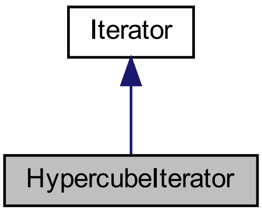
- lib/hnco/algorithms/human.hh
- lib/hnco/algorithms/human.cc

5.42 HypercubeIterator Class Reference

Hypercube iterator.

`#include <hnco/iterator.hh>`

Inheritance diagram for HypercubeIterator:



Public Member Functions

- HypercubeIterator (int n)

  Constructor.
- bool has_next ()

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

  Next bit vector.

Additional Inherited Members

5.42.1 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

5.43 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.

5.43.1 Detailed Description

Implementation

Definition at line 37 of file implementation.hh.

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

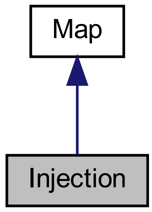
- lib/hnco/algorithms/fast-efficient-p3/implementation.hh

5.44 Injection Class Reference

Injection.

`#include <hnco/map.hh>`

Inheritance diagram for Injection:



Public Member Functions

- Injection (const std::vector< int > &bit_positions, int output_size)
  Constructor.
- void map (const bit_vector_t &input, bit_vector_t &output)
  Map
- int get_input_size ()
  Get input size.
- int get_output_size ()
  Get output size.
- bool is_surjective ()
  Check for surjective map.

Private Attributes

- std::vector< int > _bit_positions
  Bit positions.
- int _output_size
  Output size.

5.44.1 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 403 of file map.hh.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 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 144 of file map.cc.

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

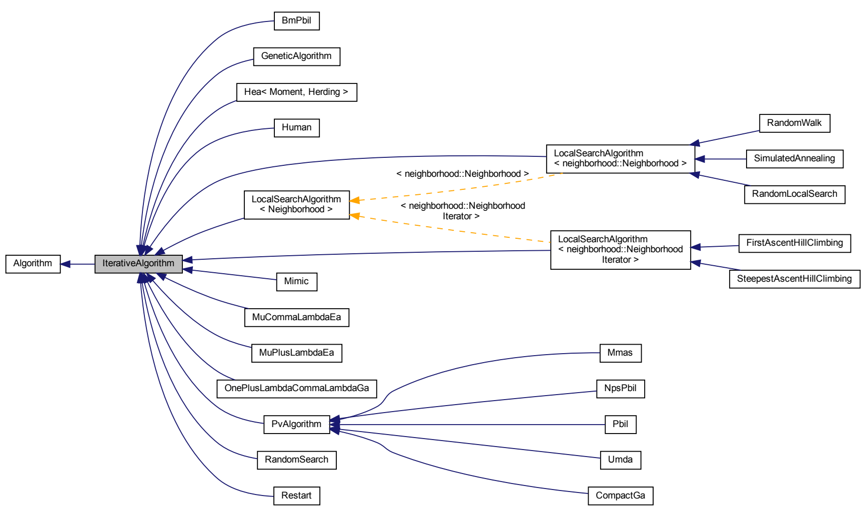
- lib/hnco/map.hh
- lib/hnco/map.cc

5.45 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)
  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 ()
  Loop.

Protected Attributes

- int _iteration
  Current iteration.
- bool _something_to_log = false
  Something to log.

Parameters

- int _num_iterations = 0
  Number of iterations.

5.45.1 Detailed Description

Iterative search.

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

5.45.2 Constructor & Destructor Documentation

5.45.2.1 IterativeAlgorithm()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 76 of file iterative-algorithm.hh.

5.45.3 Member Function Documentation

5.45.3.1 maximize()

```
void maximize (
            const std::vector< function::Function * > & functions )  [virtual]
```

Maximize.

It is essentially a loop which, at each iteration, calls iterate() then log() only if _something_to_log is true.

Warning

If an exception such as LocalMaximumReached is thrown by iterate(), log() will not be called.

Implements Algorithm.

Definition at line 48 of file iterative-algorithm.cc.

5.45.3.2 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 103 of file iterative-algorithm.hh.

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

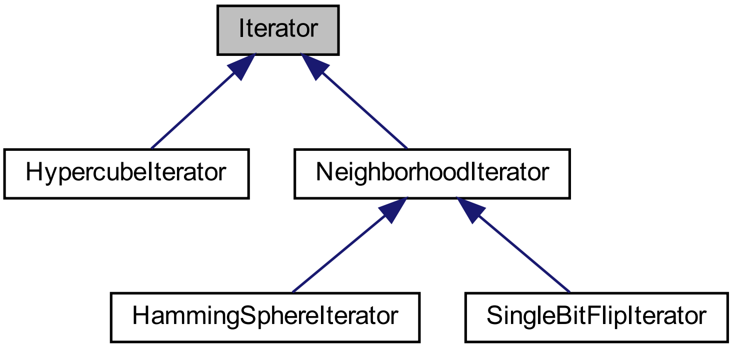
- lib/hnco/algorithms/iterative-algorithm.hh
- lib/hnco/algorithms/iterative-algorithm.cc

5.46 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.

5.46.1 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:

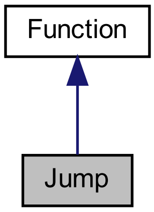
- lib/hnco/iterator.hh

5.47 Jump Class Reference

Jump.

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

Inheritance diagram for Jump:



Public Member Functions

- Jump (int bv_size, int gap)

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- bool has_known_maximum ()

  Check for a known maximum.
- double get_maximum ()

  Get the global maximum.

Private Attributes

- int _bv_size

  Bit vector size.
- int _gap

  Gap.

5.47.1 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.

5.47.2 Member Function Documentation

5.47.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 67 of file jump.hh.

5.47.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 63 of file jump.hh.

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

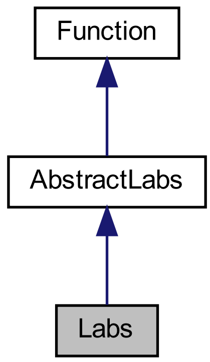
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

5.48 Labs Class Reference

Low autocorrelation binary sequences.

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

Inheritance diagram for Labs:



Public Member Functions

- Labs (int n)

  Constructor.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Additional Inherited Members

5.48.1 Detailed Description

Low autocorrelation binary sequences.

Reference:

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

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

Definition at line 65 of file labs.hh.

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

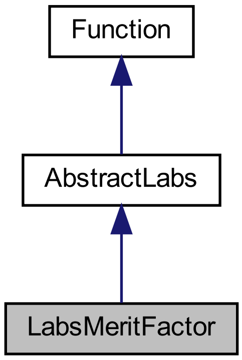
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.49 LabsMeritFactor Class Reference

Low autocorrelation binary sequences merit factor.

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

Inheritance diagram for LabsMeritFactor:



Public Member Functions

- LabsMeritFactor (int n)

  Constructor.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Additional Inherited Members

5.49.1 Detailed Description

Low autocorrelation binary sequences merit factor.

Reference:

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

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

Definition at line 90 of file labs.hh.

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

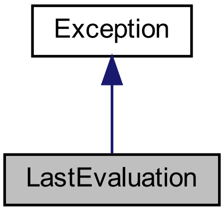
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

5.50 LastEvaluation Class Reference

Last evaluation.

`#include <hnco/exception.hh>`

Inheritance diagram for LastEvaluation:



5.50.1 Detailed Description

Last evaluation.

Definition at line 93 of file exception.hh.

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

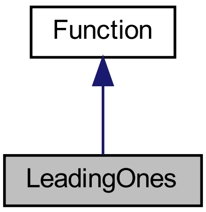
- lib/hnco/exception.hh

5.51 LeadingOnes Class Reference

Leading ones.

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

Inheritance diagram for LeadingOnes:



Public Member Functions

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

Private Attributes

- int _bv_size
  Bit vector size.

5.51.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 98 of file theory.hh.

5.51.2 Member Function Documentation

5.51.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 122 of file theory.hh.

5.51.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 118 of file theory.hh.

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

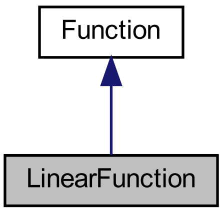
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.52 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



Public Member Functions

- LinearFunction ()

  Constructor.

Instance generators

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

  Instance generator.
- void random (int n)

  Random instance.

Evaluation

- double evaluate (const bit_vector_t &)

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

  Incrementally evaluate a bit vector.

Information about the function

- int get_bv_size ()

  Get bit vector size.
- double get_maximum ()

  Get the global maximum.
- bool has_known_maximum ()

  Check for a known maximum.
- bool provides_incremental_evaluation ()

  Check whether the function provides incremental evaluation.

Private Member Functions

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

  Serialize.

Private Attributes

- std::vector< double > _weights

  Weights.

Friends

- class **boost::serialization::access**

5.52.1 Detailed Description

Linear function.

Definition at line 42 of file linear-function.hh.

5.52.2 Member Function Documentation

5.52.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

Definition at line 73 of file linear-function.hh.

5.52.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 117 of file linear-function.hh.

5.52.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

5.52.2.4 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 85 of file linear-function.hh.

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

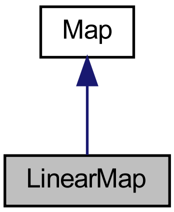
- lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

5.53 LinearMap Class Reference

Linear map.

`#include <hnco/map.hh>`

Inheritance diagram for LinearMap:



Public Member Functions

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

  Random instance.
- void map (const bit_vector_t &input, bit_vector_t &output)

  Map
- int get_input_size ()

  Get input size.
- int get_output_size ()

  Get output size.
- bool is_surjective ()

  Check for surjective map.

Private Member Functions

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

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

  Load.

Private Attributes

- bit_matrix_t _bm

  Bit matrix.

Friends

- class **boost::serialization::access**

5.53.1 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 200 of file map.hh.

5.53.2 Member Function Documentation

5.53.2.1 is_surjective()

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

Check for surjective map.

Returns

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

Reimplemented from Map.

Definition at line 90 of file map.cc.

5.53.2.2 random()

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

Random instance.

Parameters

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

Exceptions

Error	

Definition at line 61 of file map.cc.

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

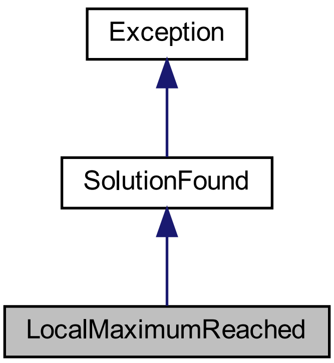
- lib/hnco/map.hh
- lib/hnco/map.cc

5.54 LocalMaximumReached Class Reference

Local maximum reached.

`#include <hnco/exception.hh>`

Inheritance diagram for LocalMaximumReached:



Public Member Functions

- LocalMaximumReached (const algorithm::solution_t &solution)

    Constructor.

Additional Inherited Members

5.54.1 Detailed Description

Local maximum reached.

Definition at line 82 of file exception.hh.

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

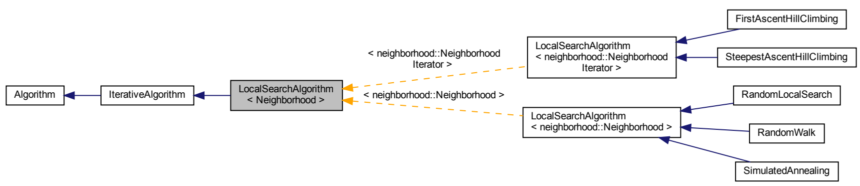
- lib/hnco/exception.hh

5.55 LocalSearchAlgorithm< Neighborhood > Class Template Reference

Local search algorithm.

`#include <hnco/algorithms/ls/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.

5.55.1 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:

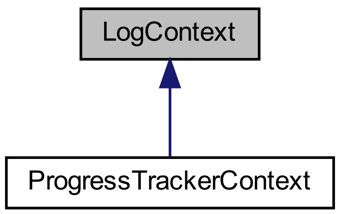
- lib/hnco/algorithms/ls/local-search-algorithm.hh

5.56 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.

5.56.1 Detailed Description

Log context.

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

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

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

- lib/hnco/logging/log-context.hh

5.57 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.

5.57.1 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.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 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.

5.57.2.2 ∼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:

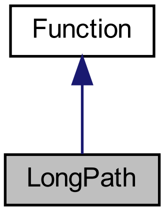
- lib/hnco/logging/logger.hh
- lib/hnco/logging/logger.cc

5.58 LongPath Class Reference

Long path.

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

Inheritance diagram for LongPath:



Public Member Functions

- LongPath (int bv_size, int prefix_length)

    Constructor.
- double evaluate (const bit_vector_t &)

    Evaluate a bit vector.

Information about the function

- int get_bv_size ()

    Get bit vector size.
- bool has_known_maximum ()

    Check for a known maximum.
- double get_maximum ()

    Get the global maximum.

Private Attributes

- int _bv_size

    Bit vector size.
- int _prefix_length

    Prefix length.

5.58.1 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.

5.58.2 Member Function Documentation

5.58.2.1 get_maximum()

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

Get the global maximum.

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

Exceptions

Error	

Reimplemented from Function.

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

5.58.2.2 has_known_maximum()

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

Check for a known maximum.

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

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

If $\log_2(k) + n/k \leq 53$ 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:

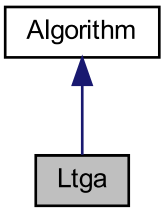
- lib/hnco/functions/long-path.hh
- lib/hnco/functions/long-path.cc

5.59 Ltga Class Reference

Linkage Tree Genetic Algorithm.

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

Inheritance diagram for Ltga:



Public Member Functions

- Ltga (int n)

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

  Maximize.
- void finalize ()

  Finalize.
- void set_population_size (int n)

  Set population size.

Private Attributes

- std::unique_ptr< Implementation > _pimpl

  Pointer to implementation.
- int _population_size = 10

  Population size.

Additional Inherited Members

5.59.1 Detailed Description

Linkage Tree Genetic Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 47 of file ltga.hh.

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

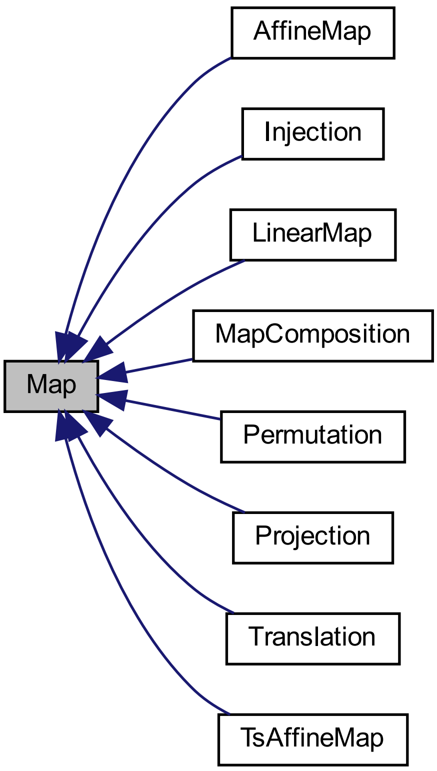
- lib/hnco/algorithms/fast-efficient-p3/ltga.hh
- lib/hnco/algorithms/fast-efficient-p3/ltga.cc

5.60 Map Class Reference

Map

`#include <hnco/map.hh>`

Inheritance diagram for Map:



Public Member Functions

- virtual ∼Map ()
  Destructor.
- virtual void map (const bit_vector_t &input, bit_vector_t &output)=0
  Map
- virtual int get_input_size ()=0
  Get input size.
- virtual int get_output_size ()=0
  Get output size.
- virtual bool is_surjective ()
  Check for surjective map.
- virtual void display (std::ostream &stream)
  Display.

5.60.1 Detailed Description

Map

Definition at line 40 of file map.hh.

5.60.2 Member Function Documentation

5.60.2.1 is_surjective()

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

Check for surjective map.

Returns

false

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

Definition at line 60 of file map.hh.

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

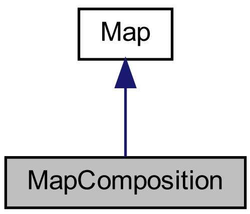
- lib/hnco/map.hh

5.61 MapComposition Class Reference

Map composition.

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

Inheritance diagram for MapComposition:



Public Member Functions

- MapComposition ()
  Default constructor.
- MapComposition (Map ∗outer, Map ∗inner)
  Constructor.
- void map (const bit_vector_t &input, bit_vector_t &output)
  Map
- int get_input_size ()
  Get input size.
- int get_output_size ()
  Get output size.
- bool is_surjective ()
  Check for surjective map.

Private Attributes

- Map ∗ _outer
  Outer map.
- Map ∗ _inner
  Inner map.
- bit_vector_t _bv
  Temporary bit vector.

5.61.1 Detailed Description

Map composition.

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

Definition at line 334 of file map.hh.

5.61.2 Constructor & Destructor Documentation

5.61.2.1 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 358 of file map.hh.

5.61.3 Member Function Documentation

5.61.3.1 is_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 382 of file map.hh.

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

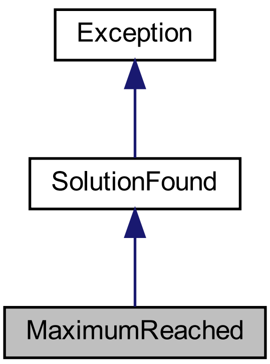
- lib/hnco/map.hh

5.62 MaximumReached Class Reference

Maximum reached.

`#include <hnco/exception.hh>`

Inheritance diagram for MaximumReached:



Public Member Functions

- MaximumReached (const algorithm::solution_t &solution)

  Constructor.

Additional Inherited Members

5.62.1 Detailed Description

Maximum reached.

Definition at line 60 of file exception.hh.

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

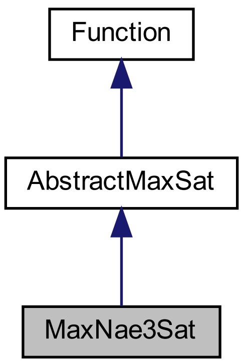
- lib/hnco/exception.hh

5.63 MaxNae3Sat Class Reference

Max not-all-equal 3SAT.

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

Inheritance diagram for MaxNae3Sat:



Public Member Functions

- MaxNae3Sat ()

  Default constructor.
- void load (std::istream &stream)

  Load an instance.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Additional Inherited Members

5.63.1 Detailed Description

Max not-all-equal 3SAT.

Reference:

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

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

5.63.2 Member Function Documentation

5.63.2.1 load()

```
void load (
            std::istream & stream )  [virtual]
```

Load an instance.

Exceptions

Error	

Reimplemented from AbstractMaxSat.

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

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

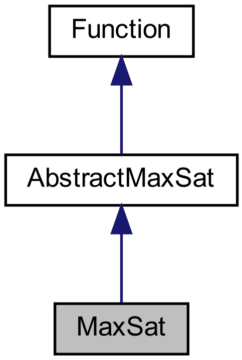
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.64 MaxSat Class Reference

MAX-SAT.

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

Inheritance diagram for MaxSat:



Public Member Functions

- MaxSat ()

  Default constructor.
- void random (int n, int k, int c)

  Random instance.
- void random (const bit_vector_t &solution, int k, int c)

  Random instance with satisfiable expression.
- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Additional Inherited Members

5.64.1 Detailed Description

MAX-SAT.

Reference:

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

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

5.64.2 Member Function Documentation

5.64.2.1 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.

5.64.2.2 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:

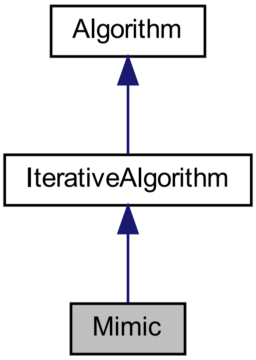
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

5.65 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.

5.65.1 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:

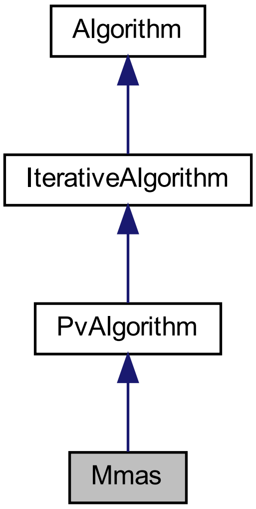
- lib/hnco/algorithms/mimic.hh
- lib/hnco/algorithms/mimic.cc

5.66 Mmas Class Reference

Max-min ant system.

`#include <hnco/algorithms/pv/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.

5.66.1 Detailed Description

Max-min ant system.

Reference:

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

Definition at line 42 of file mmas.hh.

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

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

5.67 Model Class Reference

Model of a Boltzmann machine

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

Public Member Functions

- Model (int n)
  Constructor.
- void init ()
  Initialize.
- void reset_mc ()
  Reset Markov chain.
- void gibbs_sampler (int i)
  A Gibbs sampler cycle.
- void gibbs_sampler_synchronous ()
  A synchronous Gibbs sampler.
- const bit_vector_t & get_state ()
  Get the state of the Gibbs sampler.
- void update (const ModelParameters &p, const ModelParameters &q, double rate)
  Update parameters in the direction of p and away from q.
- double norm_infinite ()
  Infinite norm of the parameters.
- double norm_l1 ()
  l1 norm of the parameters

Private Attributes

- ModelParameters _model_parameters

  Model parameters.
- bit_vector_t _state

  State of the Gibbs sampler.
- pv_t _pv

  Probability vector for synchronous Gibbs sampling.

5.67.1 Detailed Description

Model of a Boltzmann machine

Definition at line 102 of file model.hh.

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

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

5.68 ModelParameters Class Reference

Parameters of a Boltzmann machine.

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

Public Member Functions

- ModelParameters (int n)

  Constructor.
- void init ()

  Initialize.
- void add (const bit_vector_t &x)

  Add a bit vector.
- void average (int count)

  Compute averages.
- void update (const ModelParameters &p, const ModelParameters &q, double rate)

  Update parameters in the direction of p and away from q.
- double norm_infinite ()

  Infinite norm of the parameters.
- double norm_l1 ()

  l1 norm of the parameters

Private Attributes

- std::vector< std::vector< double > > _weight

  Weights.
- std::vector< double > _bias

  Bias.

Friends

- class **Model**

5.68.1 Detailed Description

Parameters of a Boltzmann machine.

Definition at line 36 of file model.hh.

5.68.2 Member Function Documentation

5.68.2.1 add()

```
void add (
            const bit_vector_t & x )
```

Add a bit vector.

Only the upper triangular part of _weight is updated with the equation:

$w_{ij} = w_{ij} + (-1)^{x_i + x_j}$

where i $<$ j.

Definition at line 47 of file model.cc.

5.68.2.2 average()

```
void average (
            int count )
```

Compute averages.

Only the upper triangular part of _weight is averaged.

Definition at line 72 of file model.cc.

5.68.2.3 init()

```
void init ( )
```

Initialize.

All entries of _weight are set to 0.

Definition at line 38 of file model.cc.

5.68.2.4 update()

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

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

First, the upper triangular part of _weight is updated.

Second, _weight is made symmetrical.

Postcondition

_weight is symmetrical.

Definition at line 84 of file model.cc.

5.68.3 Member Data Documentation

5.68.3.1 _weight

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

Weights.

_weight is a full square matrix of order n, where n is the dimension of the search space.

Definition at line 43 of file model.hh.

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

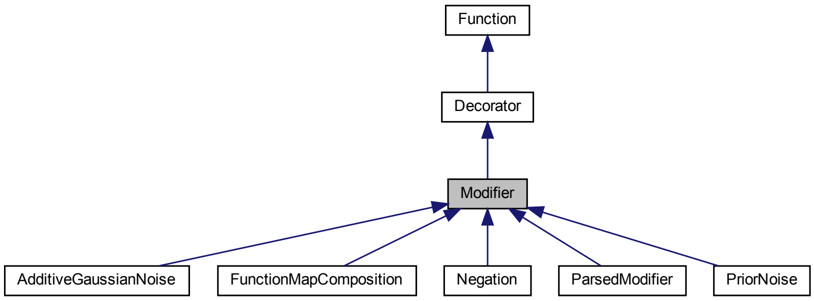
- lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

5.69 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

5.69.1 Detailed Description

Function modifier.

Definition at line 39 of file modifier.hh.

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

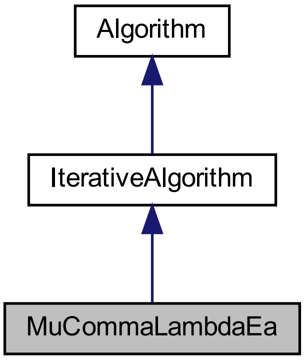
- lib/hnco/functions/modifiers/modifier.hh

5.70 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



Public Member Functions

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

  Constructor.

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.
- 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.

5.70.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.70.2 Constructor & Destructor Documentation

5.70.2.1 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 89 of file mu-comma-lambda-ea.hh.

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

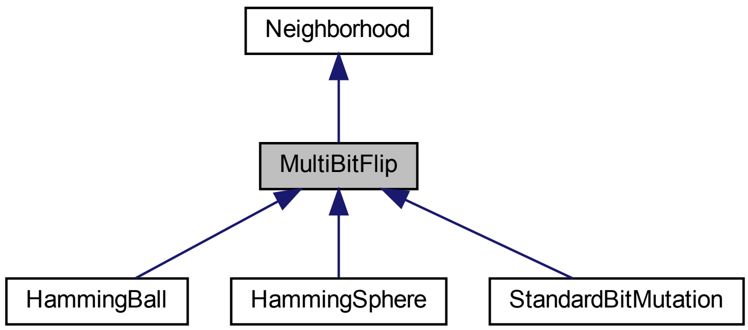
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

5.71 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

5.71.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

5.71.2 Constructor & Destructor Documentation

5.71.2.1 MultiBitFlip()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 206 of file neighborhood.hh.

5.71.3 Member Function Documentation

5.71.3.1 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.

5.71.3.2 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:

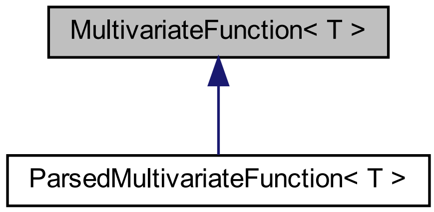
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.72 MultivariateFunction< T > Class Template Reference

Multivariate function.

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

Inheritance diagram for MultivariateFunction< T >:



Public Types

- typedef T value_type
  Type of value.

Public Member Functions

- virtual ~MultivariateFunction ()
  Destructor.
- virtual int get_num_variables ()=0
  Get the number of variables.
- virtual T evaluate (const std::vector< value_type > &x)=0
  Evaluate.

5.72.1 Detailed Description

template<class T>
class hnco::function::representation::MultivariateFunction< T >

Multivariate function.

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

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

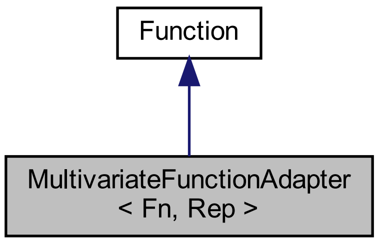
- lib/hnco/functions/representations/multivariate-function.hh

5.73 MultivariateFunctionAdapter< Fn, Rep > Class Template Reference

Multivariate function adapter.

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

Inheritance diagram for MultivariateFunctionAdapter< Fn, Rep >:



Public Member Functions

- MultivariateFunctionAdapter (Fn *fn, std::vector< Rep > reps)

    Constructor.

Information about the function

- int get_bv_size () override

    Get bit vector size.

Evaluation

- double evaluate (const bit_vector_t &bv) override

    Evaluate.

Display

- void display (std::ostream &stream) 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 Rep::value_type > _variables

    Variables.

5.73.1 Detailed Description

template<class Fn, class Rep>
class hnco::function::representation::MultivariateFunctionAdapter< Fn, Rep >

Multivariate function adapter.

Definition at line 43 of file multivariate-function-adapter.hh.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 MultivariateFunctionAdapter()

```
MultivariateFunctionAdapter (
            Fn * fn,
            std::vector< Rep > reps )  [inline]
```

Constructor.

Parameters

fn	Multivariate function
reps	Representations

Definition at line 75 of file multivariate-function-adapter.hh.

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

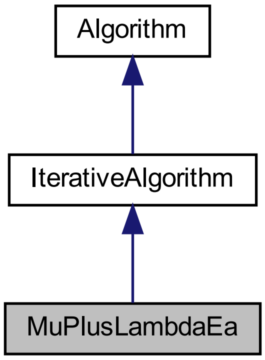
- lib/hnco/functions/representations/multivariate-function-adapter.hh

5.74 MuPlusLambdaEa Class Reference

(mu+lambda) EA.

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

Inheritance diagram for MuPlusLambdaEa:



Public Member Functions

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

  Constructor.

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.
- 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.

5.74.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

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

5.74.2 Constructor & Destructor Documentation

5.74.2.1 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 89 of file mu-plus-lambda-ea.hh.

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

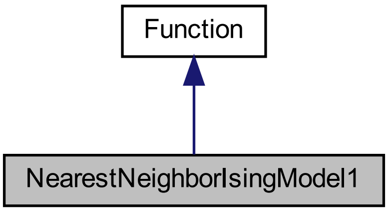
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

5.75 NearestNeighborIsingModel1 Class Reference

Nearest neighbor Ising model in one dimension.

```
#include <hnco/functions/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.
- void display (std::ostream &stream)
  Display.

Instance generators

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

Evaluation

- double evaluate (const bit_vector_t &)
  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.

Information about the function

- int get_bv_size ()
  Get bit vector size.
- bool provides_incremental_evaluation ()
  Check whether the function provides incremental evaluation.

Private Member Functions

- template<class Archive >
  void save (Archive &ar, const unsigned int version) const
  Save.
- template<class Archive >
  void load (Archive &ar, const unsigned int version)
  Load.
- void resize (int n)
  Resize data structures.

Private Attributes

- std::vector< double > _coupling

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

  External field.
- bit_vector_t _flipped_bits

  Flipped bits.
- bool _periodic_boundary_conditions = false

  Periodic boundary conditions.

Friends

- class **boost::serialization::access**

5.75.1 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 65 of file nearest-neighbor-ising-model-1.hh.

5.75.2 Member Function Documentation

5.75.2.1 evaluate()

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

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.75.2.2 generate()

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

Instance generator.

Parameters

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

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

5.75.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

5.75.2.4 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 142 of file nearest-neighbor-ising-model-1.hh.

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

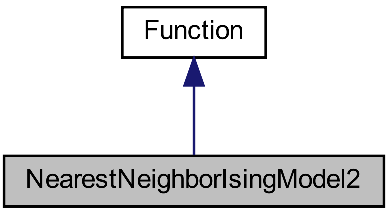
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-1.cc

5.76 NearestNeighborIsingModel2 Class Reference

Nearest neighbor Ising model in two dimensions.

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

Inheritance diagram for NearestNeighborIsingModel2:



Public Member Functions

- NearestNeighborIsingModel2 ()
  Constructor.
- void set_periodic_boundary_conditions (bool x)
  Set periodic boundary conditions.
- void display (std::ostream &stream)
  Display.

Instance generators

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

Evaluation

- double evaluate (const bit_vector_t &)
  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double v, const sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.

Information about the function

- int get_bv_size ()
  Get bit vector size.
- bool provides_incremental_evaluation ()
  Check whether the function provides incremental evaluation.

Private Member Functions

- template<class Archive >
  void save (Archive &ar, const unsigned int version) const
  Save.
- template<class Archive >
  void load (Archive &ar, const unsigned int version)
  Load.
- void resize (int num_rows, int num_columns)
  Resize data structures.

Private Attributes

- std::vector< std::vector< double > > _coupling_right
  Coupling with nearest neighbor to the right.
- std::vector< std::vector< double > > _coupling_below
  Coupling with nearest neighbor below.
- std::vector< std::vector< double > > _field
  External field.
- bit_vector_t _flipped_bits
  Flipped bits.
- bool _periodic_boundary_conditions = false
  Periodic boundary conditions.

Friends

- class **boost::serialization::access**

5.76.1 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 67 of file nearest-neighbor-ising-model-2.hh.

5.76.2 Member Function Documentation

5.76.2.1 evaluate()

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

Evaluate a bit vector.

Complexity: O(n)

Implements Function.

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

5.76.2.2 generate()

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

Instance generator.

Parameters

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

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

5.76.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

5.76.2.4 random()

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

Random instance.

The weights are sampled from the normal distribution.

Parameters

num_rows	Number of rows
num_columns	Number of columns

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

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

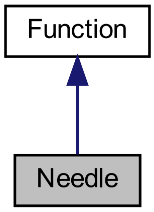
- lib/hnco/functions/ising/nearest-neighbor-ising-model-2.hh
- lib/hnco/functions/ising/nearest-neighbor-ising-model-2.cc

5.77 Needle Class Reference

Needle in a haystack.

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

Inheritance diagram for Needle:



Public Member Functions

- Needle (int bv_size)

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- bool has_known_maximum ()

  Check for a known maximum.
- double get_maximum ()

  Get the global maximum.

Private Attributes

- int _bv_size

  Bit vector size.

5.77.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 134 of file theory.hh.

5.77.2 Member Function Documentation

5.77.2.1 get_maximum()

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

Get the global maximum.

Returns

1

Reimplemented from Function.

Definition at line 158 of file theory.hh.

5.77.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 154 of file theory.hh.

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

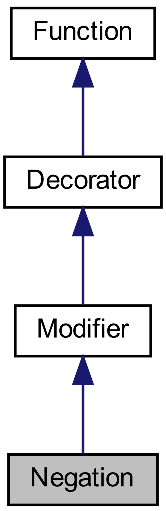
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.78 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 ()

  Get bit vector size.
- bool provides_incremental_evaluation ()

  Check whether the function provides incremental evaluation.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.

Additional Inherited Members

5.78.1 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.

5.78.2 Member Function Documentation

5.78.2.1 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 79 of file modifier.hh.

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

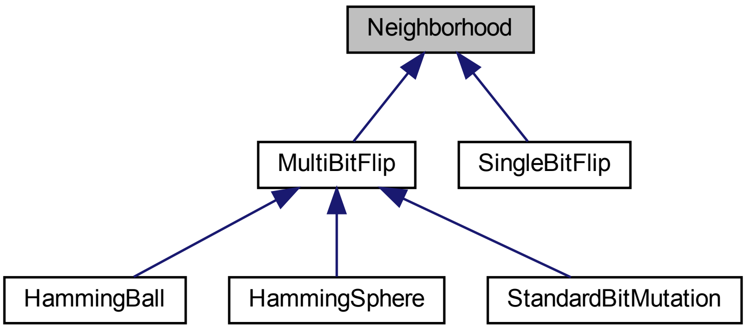
- lib/hnco/functions/modifiers/modifier.hh
- lib/hnco/functions/modifiers/modifier.cc

5.79 Neighborhood Class Reference

Neighborhood.

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

Inheritance diagram for Neighborhood:



Public Member Functions

- Neighborhood (int n)

    Constructor.
- virtual ∼Neighborhood ()

    Destructor.
- virtual void set_origin (const bit_vector_t &x)

    Set the origin.
- virtual const bit_vector_t & get_origin ()

    Get the origin.
- virtual const bit_vector_t & get_candidate ()

    Get the candidate bit vector.
- virtual const sparse_bit_vector_t & get_flipped_bits ()

    Get flipped bits.
- virtual void propose ()

    Propose a candidate bit vector.
- virtual void keep ()

    Keep the candidate bit vector.
- virtual void forget ()

    Forget the candidate bit vector.
- virtual void mutate (bit_vector_t &bv)

    Mutate.
- virtual void map (const bit_vector_t &input, bit_vector_t &output)

    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.

5.79.1 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.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 Neighborhood()

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

Constructor.

Parameters

n	Size of bit vectors

Definition at line 86 of file neighborhood.hh.

5.79.3 Member Function Documentation

5.79.3.1 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 148 of file neighborhood.hh.

5.79.3.2 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 134 of file neighborhood.hh.

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

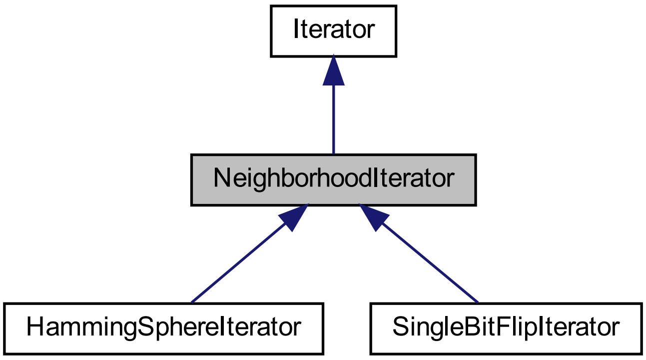
- lib/hnco/neighborhoods/neighborhood.hh

5.80 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

5.80.1 Detailed Description

Neighborhood iterator

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

5.80.2 Constructor & Destructor Documentation

5.80.2.1 NeighborhoodIterator()

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

Constructor.

Parameters

n	Size of bit vectors

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

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

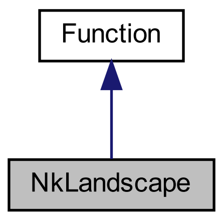
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.81 NkLandscape Class Reference

NK landscape.

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

Inheritance diagram for NkLandscape:



Public Member Functions

- NkLandscape ()

  Default constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- void display (std::ostream &stream)

  Display.

Instance generators

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

  Instance generator.
- void random (int n, int k)

  Random instance.

Private Member Functions

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

  Serialize.
- void random_structure (int n, int k)

  Random structue.

Private Attributes

- std::vector< std::vector< int > > _neighbors

  Bit neighbors.
- std::vector< std::vector< double > > _partial_functions

  Partial functions.

Friends

- class **boost::serialization::access**

5.81.1 Detailed Description

NK landscape.

Reference:

S. A. Kauffman. 1993. The origins of order: self-organisation and selection in evolution. Oxford University Press.

Definition at line 47 of file nk-landscape.hh.

5.81.2 Member Function Documentation

5.81.2.1 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 92 of file nk-landscape.hh.

5.81.2.2 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 110 of file nk-landscape.hh.

5.81.2.3 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.

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

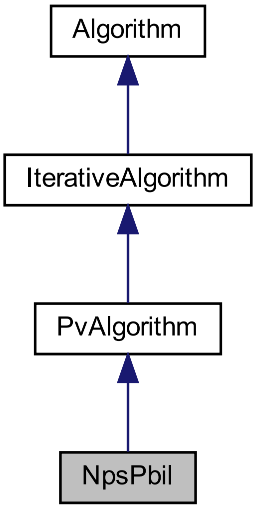
- lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

5.82 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

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

Inheritance diagram for NpsPbil:



Public Member Functions

- NpsPbil (int n, int population_size)

  Constructor.

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.

5.82.1 Detailed Description

Population-based incremental learning with negative and positive selection.

Reference:

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

Definition at line 42 of file nps-pbil.hh.

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

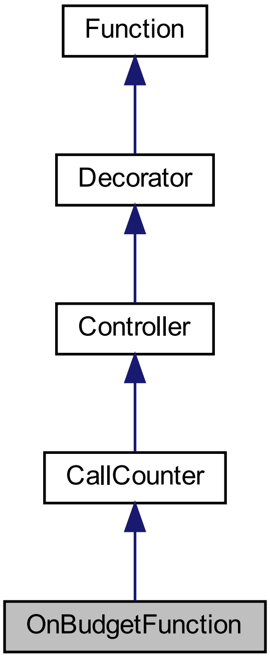
- lib/hnco/algorithms/pv/nps-pbil.hh
- lib/hnco/algorithms/pv/nps-pbil.cc

5.83 OnBudgetFunction Class Reference

CallCounter 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 &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &x, double value)

  Update after a safe evaluation.

Private Attributes

- int _budget

  Budget.

Additional Inherited Members

5.83.1 Detailed Description

CallCounter with a limited number of evaluations.

Definition at line 319 of file controller.hh.

5.83.2 Member Function Documentation

5.83.2.1 evaluate()

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

Evaluate a bit vector.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 123 of file controller.cc.

5.83.2.2 evaluate_incrementally()

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

Incrementally evaluate a bit vector.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 134 of file controller.cc.

5.83.2.3 update()

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

Update after a safe evaluation.

Exceptions

LastEvaluation	

Reimplemented from CallCounter.

Definition at line 145 of file controller.cc.

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

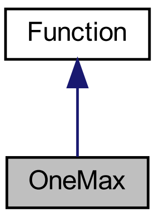
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.84 OneMax Class Reference

OneMax.

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

Inheritance diagram for OneMax:



Public Member Functions

- OneMax (int bv_size)

    Constructor.

Information about the function

- int get_bv_size ()

    Get bit vector size.
- double get_maximum ()

    Get the global maximum.
- bool has_known_maximum ()

    Check for a known maximum.
- bool provides_incremental_evaluation ()

    Check whether the function provides incremental evaluation.

Evaluation

- double evaluate (const bit_vector_t &)

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

    Incrementally evaluate a bit vector.

Private Attributes

- int _bv_size

    Bit vector size.

5.84.1 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.

5.84.2 Member Function Documentation

5.84.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 62 of file theory.hh.

5.84.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 66 of file theory.hh.

5.84.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

Definition at line 71 of file theory.hh.

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

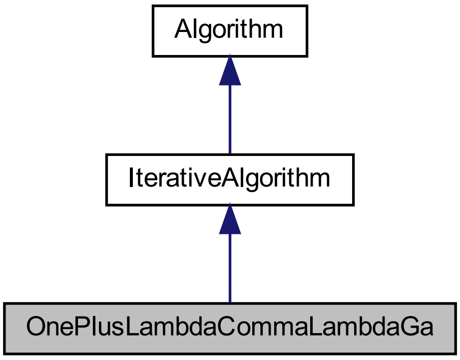
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.85 OnePlusLambdaCommaLambdaGa Class Reference

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

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

Inheritance diagram for OnePlusLambdaCommaLambdaGa:



Public Member Functions

- OnePlusLambdaCommaLambdaGa (int n, int lambda)

    Constructor.

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.

5.85.1 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.

5.85.2 Constructor & Destructor Documentation

5.85.2.1 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 103 of file one-plus-lambda-comma-lambda-ga.hh.

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

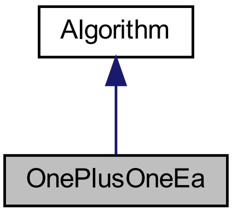
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

5.86 OnePlusOneEa Class Reference

(1+1) EA.

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

Inheritance diagram for OnePlusOneEa:



Public Member Functions

- OnePlusOneEa (int n)
  Constructor.
- void 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

5.86.1 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.

5.86.2 Constructor & Destructor Documentation

5.86.2.1 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.

5.86.3 Member Function Documentation

5.86.3.1 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/ea/one-plus-one-ea.hh

5.87 Options Class Reference

Command line options.

`#include </home/arnaud/projets/hnco/src/hnco/app/ffgen-options.hh>`

Public Member Functions

- Options (int argc, char ∗argv[ ])

  Constructor.
- int get_bv_size () const

  Get bv_size.
- void set_bv_size (int x)

  Set bv_size.
- bool set_bv_size () const

  Get set-flag for bv_size.
- double get_coupling_constant () const

  Get coupling_constant.
- void set_coupling_constant (double x)

  Set coupling_constant.
- bool set_coupling_constant () const

Get set-flag for coupling_constant.
- double get_ep_upper_bound () const

  Get ep_upper_bound.
- void set_ep_upper_bound (double x)

  Set ep_upper_bound.
- bool set_ep_upper_bound () const

  Get set-flag for ep_upper_bound.
- double get_field_constant () const

  Get field_constant.
- void set_field_constant (double x)

  Set field_constant.
- bool set_field_constant () const

  Get set-flag for field_constant.
- int get_function () const

  Get function.
- void set_function (int x)

  Set function.
- bool set_function () const

  Get set-flag for function.
- double get_lin_distance () const

  Get lin_distance.
- void set_lin_distance (double x)

  Set lin_distance.
- bool set_lin_distance () const

  Get set-flag for lin_distance.
- int get_lin_generator () const

  Get lin_generator.
- void set_lin_generator (int x)

  Set lin_generator.
- bool set_lin_generator () const

  Get set-flag for lin_generator.
- double get_lin_initial_weight () const

  Get lin_initial_weight.
- void set_lin_initial_weight (double x)

  Set lin_initial_weight.
- bool set_lin_initial_weight () const

  Get set-flag for lin_initial_weight.
- double get_lin_ratio () const

  Get lin_ratio.
- void set_lin_ratio (double x)

  Set lin_ratio.
- bool set_lin_ratio () const

  Get set-flag for lin_ratio.
- int get_ms_num_clauses () const

  Get ms_num_clauses.
- void set_ms_num_clauses (int x)

  Set ms_num_clauses.
- bool set_ms_num_clauses () const

  Get set-flag for ms_num_clauses.
- int get_ms_num_literals_per_clause () const

  Get ms_num_literals_per_clause.

- void set_ms_num_literals_per_clause (int x)

    Set ms_num_literals_per_clause.
- bool set_ms_num_literals_per_clause () const

    Get set-flag for ms_num_literals_per_clause.
- int get_nk_k () const

    Get nk_k.
- void set_nk_k (int x)

    Set nk_k.
- bool set_nk_k () const

    Get set-flag for nk_k.
- int get_nn1_generator () const

    Get nn1_generator.
- void set_nn1_generator (int x)

    Set nn1_generator.
- bool set_nn1_generator () const

    Get set-flag for nn1_generator.
- int get_nn2_generator () const

    Get nn2_generator.
- void set_nn2_generator (int x)

    Set nn2_generator.
- bool set_nn2_generator () const

    Get set-flag for nn2_generator.
- int get_nn2_num_columns () const

    Get nn2_num_columns.
- void set_nn2_num_columns (int x)

    Set nn2_num_columns.
- bool set_nn2_num_columns () const

    Get set-flag for nn2_num_columns.
- int get_nn2_num_rows () const

    Get nn2_num_rows.
- void set_nn2_num_rows (int x)

    Set nn2_num_rows.
- bool set_nn2_num_rows () const

    Get set-flag for nn2_num_rows.
- int get_part_upper_bound () const

    Get part_upper_bound.
- void set_part_upper_bound (int x)

    Set part_upper_bound.
- bool set_part_upper_bound () const

    Get set-flag for part_upper_bound.
- std::string get_path () const

    Get path.
- void set_path (std::string x)

    Set path.
- bool set_path () const

    Get set-flag for path.
- int get_seed () const

    Get seed.
- void set_seed (int x)

    Set seed.
- bool set_seed () const

Get set-flag for seed.

- double get_stddev () const

  Get stddev.
- void set_stddev (double x)

  Set stddev.
- bool set_stddev () const

  Get set-flag for stddev.
- int get_walsh2_generator () const

  Get walsh2_generator.
- void set_walsh2_generator (int x)

  Set walsh2_generator.
- bool set_walsh2_generator () const

  Get set-flag for walsh2_generator.
- double get_walsh2_ising_alpha () const

  Get walsh2_ising_alpha.
- void set_walsh2_ising_alpha (double x)

  Set walsh2_ising_alpha.
- bool set_walsh2_ising_alpha () const

  Get set-flag for walsh2_ising_alpha.
- int get_walsh_num_features () const

  Get walsh_num_features.
- void set_walsh_num_features (int x)

  Set walsh_num_features.
- bool set_walsh_num_features () const

  Get set-flag for walsh_num_features.
- bool with_ms_planted_solution () const

  Get ms_planted_solution.
- void set_ms_planted_solution ()

  Set ms_planted_solution.
- bool with_periodic_boundary_conditions () const

  Get periodic_boundary_conditions.
- void set_periodic_boundary_conditions ()

  Set periodic_boundary_conditions.
- Options (int argc, char ∗argv[ ])

  Constructor.
- int get_algorithm () const

  Get algorithm.
- void set_algorithm (int x)

  Set algorithm.
- bool set_algorithm () const

  Get set-flag for algorithm.
- int get_bm_mc_reset_strategy () const

  Get bm_mc_reset_strategy.
- void set_bm_mc_reset_strategy (int x)

  Set bm_mc_reset_strategy.
- bool set_bm_mc_reset_strategy () const

  Get set-flag for bm_mc_reset_strategy.
- int get_bm_num_gs_cycles () const

  Get bm_num_gs_cycles.
- void set_bm_num_gs_cycles (int x)

  Set bm_num_gs_cycles.

- bool set_bm_num_gs_cycles () const

  Get set-flag for bm_num_gs_cycles.
- int get_bm_num_gs_steps () const

  Get bm_num_gs_steps.
- void set_bm_num_gs_steps (int x)

  Set bm_num_gs_steps.
- bool set_bm_num_gs_steps () const

  Get set-flag for bm_num_gs_steps.
- int get_bm_sampling () const

  Get bm_sampling.
- void set_bm_sampling (int x)

  Set bm_sampling.
- bool set_bm_sampling () const

  Get set-flag for bm_sampling.
- int get_budget () const

  Get budget.
- void set_budget (int x)

  Set budget.
- bool set_budget () const

  Get set-flag for budget.
- int get_bv_size () const

  Get bv_size.
- void set_bv_size (int x)

  Set bv_size.
- bool set_bv_size () const

  Get set-flag for bv_size.
- std::string get_description_path () const

  Get description_path.
- void set_description_path (std::string x)

  Set description_path.
- bool set_description_path () const

  Get set-flag for description_path.
- int get_ea_lambda () const

  Get ea_lambda.
- void set_ea_lambda (int x)

  Set ea_lambda.
- bool set_ea_lambda () const

  Get set-flag for ea_lambda.
- int get_ea_mu () const

  Get ea_mu.
- void set_ea_mu (int x)

  Set ea_mu.
- bool set_ea_mu () const

  Get set-flag for ea_mu.
- std::string get_expression () const

  Get expression.
- void set_expression (std::string x)

  Set expression.
- bool set_expression () const

  Get set-flag for expression.
- std::string get_fn_name () const

Get fn_name.
- void set_fn_name (std::string x)

  Set fn_name.
- bool set_fn_name () const

  Get set-flag for fn_name.
- int get_fn_num_traps () const

  Get fn_num_traps.
- void set_fn_num_traps (int x)

  Set fn_num_traps.
- bool set_fn_num_traps () const

  Get set-flag for fn_num_traps.
- int get_fn_prefix_length () const

  Get fn_prefix_length.
- void set_fn_prefix_length (int x)

  Set fn_prefix_length.
- bool set_fn_prefix_length () const

  Get set-flag for fn_prefix_length.
- int get_fn_threshold () const

  Get fn_threshold.
- void set_fn_threshold (int x)

  Set fn_threshold.
- bool set_fn_threshold () const

  Get set-flag for fn_threshold.
- std::string get_fp_expression () const

  Get fp_expression.
- void set_fp_expression (std::string x)

  Set fp_expression.
- bool set_fp_expression () const

  Get set-flag for fp_expression.
- double get_fp_lower_bound () const

  Get fp_lower_bound.
- void set_fp_lower_bound (double x)

  Set fp_lower_bound.
- bool set_fp_lower_bound () const

  Get set-flag for fp_lower_bound.
- int get_fp_num_bits () const

  Get fp_num_bits.
- void set_fp_num_bits (int x)

  Set fp_num_bits.
- bool set_fp_num_bits () const

  Get set-flag for fp_num_bits.
- double get_fp_upper_bound () const

  Get fp_upper_bound.
- void set_fp_upper_bound (double x)

  Set fp_upper_bound.
- bool set_fp_upper_bound () const

  Get set-flag for fp_upper_bound.
- int get_function () const

  Get function.
- void set_function (int x)

  Set function.

- bool set_function () const
  Get set-flag for function.
- double get_ga_crossover_bias () const
  Get ga_crossover_bias.
- void set_ga_crossover_bias (double x)
  Set ga_crossover_bias.
- bool set_ga_crossover_bias () const
  Get set-flag for ga_crossover_bias.
- double get_ga_crossover_probability () const
  Get ga_crossover_probability.
- void set_ga_crossover_probability (double x)
  Set ga_crossover_probability.
- bool set_ga_crossover_probability () const
  Get set-flag for ga_crossover_probability.
- int get_ga_tournament_size () const
  Get ga_tournament_size.
- void set_ga_tournament_size (int x)
  Set ga_tournament_size.
- bool set_ga_tournament_size () const
  Get set-flag for ga_tournament_size.
- int get_hea_bit_herding () const
  Get hea_bit_herding.
- void set_hea_bit_herding (int x)
  Set hea_bit_herding.
- bool set_hea_bit_herding () const
  Get set-flag for hea_bit_herding.
- int get_hea_num_seq_updates () const
  Get hea_num_seq_updates.
- void set_hea_num_seq_updates (int x)
  Set hea_num_seq_updates.
- bool set_hea_num_seq_updates () const
  Get set-flag for hea_num_seq_updates.
- int get_hea_reset_period () const
  Get hea_reset_period.
- void set_hea_reset_period (int x)
  Set hea_reset_period.
- bool set_hea_reset_period () const
  Get set-flag for hea_reset_period.
- int get_hea_sampling_method () const
  Get hea_sampling_method.
- void set_hea_sampling_method (int x)
  Set hea_sampling_method.
- bool set_hea_sampling_method () const
  Get set-flag for hea_sampling_method.
- double get_hea_weight () const
  Get hea_weight.
- void set_hea_weight (double x)
  Set hea_weight.
- bool set_hea_weight () const
  Get set-flag for hea_weight.
- double get_learning_rate () const

Get learning_rate.
- void set_learning_rate (double x)

    Set learning_rate.
- bool set_learning_rate () const

    Get set-flag for learning_rate.
- int get_map () const

    Get map.
- void set_map (int x)

    Set map.
- bool set_map () const

    Get set-flag for map.
- int get_map_input_size () const

    Get map_input_size.
- void set_map_input_size (int x)

    Set map_input_size.
- bool set_map_input_size () const

    Get set-flag for map_input_size.
- std::string get_map_path () const

    Get map_path.
- void set_map_path (std::string x)

    Set map_path.
- bool set_map_path () const

    Get set-flag for map_path.
- int get_map_ts_length () const

    Get map_ts_length.
- void set_map_ts_length (int x)

    Set map_ts_length.
- bool set_map_ts_length () const

    Get set-flag for map_ts_length.
- int get_map_ts_sampling_mode () const

    Get map_ts_sampling_mode.
- void set_map_ts_sampling_mode (int x)

    Set map_ts_sampling_mode.
- bool set_map_ts_sampling_mode () const

    Get set-flag for map_ts_sampling_mode.
- double get_mutation_rate () const

    Get mutation_rate.
- void set_mutation_rate (double x)

    Set mutation_rate.
- bool set_mutation_rate () const

    Get set-flag for mutation_rate.
- int get_neighborhood () const

    Get neighborhood.
- void set_neighborhood (int x)

    Set neighborhood.
- bool set_neighborhood () const

    Get set-flag for neighborhood.
- int get_neighborhood_iterator () const

    Get neighborhood_iterator.
- void set_neighborhood_iterator (int x)

    Set neighborhood_iterator.

- bool set_neighborhood_iterator () const
  Get set-flag for neighborhood_iterator.
- double get_noise_stddev () const
  Get noise_stddev.
- void set_noise_stddev (double x)
  Set noise_stddev.
- bool set_noise_stddev () const
  Get set-flag for noise_stddev.
- int get_num_iterations () const
  Get num_iterations.
- void set_num_iterations (int x)
  Set num_iterations.
- bool set_num_iterations () const
  Get set-flag for num_iterations.
- int get_num_threads () const
  Get num_threads.
- void set_num_threads (int x)
  Set num_threads.
- bool set_num_threads () const
  Get set-flag for num_threads.
- std::string get_path () const
  Get path.
- void set_path (std::string x)
  Set path.
- bool set_path () const
  Get set-flag for path.
- double get_pn_mutation_rate () const
  Get pn_mutation_rate.
- void set_pn_mutation_rate (double x)
  Set pn_mutation_rate.
- bool set_pn_mutation_rate () const
  Get set-flag for pn_mutation_rate.
- int get_pn_neighborhood () const
  Get pn_neighborhood.
- void set_pn_neighborhood (int x)
  Set pn_neighborhood.
- bool set_pn_neighborhood () const
  Get set-flag for pn_neighborhood.
- int get_pn_radius () const
  Get pn_radius.
- void set_pn_radius (int x)
  Set pn_radius.
- bool set_pn_radius () const
  Get set-flag for pn_radius.
- int get_population_size () const
  Get population_size.
- void set_population_size (int x)
  Set population_size.
- bool set_population_size () const
  Get set-flag for population_size.
- int get_pv_log_num_components () const

Get pv_log_num_components.
- void set_pv_log_num_components (int x)

  Set pv_log_num_components.
- bool set_pv_log_num_components () const

  Get set-flag for pv_log_num_components.
- int get_radius () const

  Get radius.
- void set_radius (int x)

  Set radius.
- bool set_radius () const

  Get set-flag for radius.
- std::string get_results_path () const

  Get results_path.
- void set_results_path (std::string x)

  Set results_path.
- bool set_results_path () const

  Get set-flag for results_path.
- int get_rls_patience () const

  Get rls_patience.
- void set_rls_patience (int x)

  Set rls_patience.
- bool set_rls_patience () const

  Get set-flag for rls_patience.
- double get_sa_beta_ratio () const

  Get sa_beta_ratio.
- void set_sa_beta_ratio (double x)

  Set sa_beta_ratio.
- bool set_sa_beta_ratio () const

  Get set-flag for sa_beta_ratio.
- double get_sa_initial_acceptance_probability () const

  Get sa_initial_acceptance_probability.
- void set_sa_initial_acceptance_probability (double x)

  Set sa_initial_acceptance_probability.
- bool set_sa_initial_acceptance_probability () const

  Get set-flag for sa_initial_acceptance_probability.
- int get_sa_num_transitions () const

  Get sa_num_transitions.
- void set_sa_num_transitions (int x)

  Set sa_num_transitions.
- bool set_sa_num_transitions () const

  Get set-flag for sa_num_transitions.
- int get_sa_num_trials () const

  Get sa_num_trials.
- void set_sa_num_trials (int x)

  Set sa_num_trials.
- bool set_sa_num_trials () const

  Get set-flag for sa_num_trials.
- unsigned get_seed () const

  Get seed.
- void set_seed (unsigned x)

  Set seed.

- bool set_seed () const

  Get set-flag for seed.
- int get_selection_size () const

  Get selection_size.
- void set_selection_size (int x)

  Set selection_size.
- bool set_selection_size () const

  Get set-flag for selection_size.
- std::string get_solution_path () const

  Get solution_path.
- void set_solution_path (std::string x)

  Set solution_path.
- bool set_solution_path () const

  Get set-flag for solution_path.
- double get_target () const

  Get target.
- void set_target (double x)

  Set target.
- bool set_target () const

  Get set-flag for target.
- bool with_additive_gaussian_noise () const

  Get additive_gaussian_noise.
- void set_additive_gaussian_noise ()

  Set additive_gaussian_noise.
- bool with_allow_no_mutation () const

  Get allow_no_mutation.
- void set_allow_no_mutation ()

  Set allow_no_mutation.
- bool with_bm_log_norm_infinite () const

  Get bm_log_norm_infinite.
- void set_bm_log_norm_infinite ()

  Set bm_log_norm_infinite.
- bool with_bm_log_norm_l1 () const

  Get bm_log_norm_l1.
- void set_bm_log_norm_l1 ()

  Set bm_log_norm_l1.
- bool with_bm_negative_positive_selection () const

  Get bm_negative_positive_selection.
- void set_bm_negative_positive_selection ()

  Set bm_negative_positive_selection.
- bool with_cache () const

  Get cache.
- void set_cache ()

  Set cache.
- bool with_cache_budget () const

  Get cache_budget.
- void set_cache_budget ()

  Set cache_budget.
- bool with_concrete_solution () const

  Get concrete_solution.
- void set_concrete_solution ()

Set concrete_solution.
- bool with_fn_display () const

  Get fn_display.
- void set_fn_display ()

  Set fn_display.
- bool with_fn_get_bv_size () const

  Get fn_get_bv_size.
- void set_fn_get_bv_size ()

  Set fn_get_bv_size.
- bool with_fn_get_maximum () const

  Get fn_get_maximum.
- void set_fn_get_maximum ()

  Set fn_get_maximum.
- bool with_fn_has_known_maximum () const

  Get fn_has_known_maximum.
- void set_fn_has_known_maximum ()

  Set fn_has_known_maximum.
- bool with_fn_provides_incremental_evaluation () const

  Get fn_provides_incremental_evaluation.
- void set_fn_provides_incremental_evaluation ()

  Set fn_provides_incremental_evaluation.
- bool with_fn_walsh_transform () const

  Get fn_walsh_transform.
- void set_fn_walsh_transform ()

  Set fn_walsh_transform.
- bool with_hea_bound_moment () const

  Get hea_bound_moment.
- void set_hea_bound_moment ()

  Set hea_bound_moment.
- bool with_hea_log_delta () const

  Get hea_log_delta.
- void set_hea_log_delta ()

  Set hea_log_delta.
- bool with_hea_log_dtu () const

  Get hea_log_dtu.
- void set_hea_log_dtu ()

  Set hea_log_dtu.
- bool with_hea_log_error () const

  Get hea_log_error.
- void set_hea_log_error ()

  Set hea_log_error.
- bool with_hea_log_moment_matrix () const

  Get hea_log_moment_matrix.
- void set_hea_log_moment_matrix ()

  Set hea_log_moment_matrix.
- bool with_hea_log_selection () const

  Get hea_log_selection.
- void set_hea_log_selection ()

  Set hea_log_selection.
- bool with_hea_randomize_bit_order () const

  Get hea_randomize_bit_order.

- void set_hea_randomize_bit_order ()

  Set hea_randomize_bit_order.
- bool with_incremental_evaluation () const

  Get incremental_evaluation.
- void set_incremental_evaluation ()

  Set incremental_evaluation.
- bool with_load_solution () const

  Get load_solution.
- void set_load_solution ()

  Set load_solution.
- bool with_log_improvement () const

  Get log_improvement.
- void set_log_improvement ()

  Set log_improvement.
- bool with_map_display () const

  Get map_display.
- void set_map_display ()

  Set map_display.
- bool with_map_random () const

  Get map_random.
- void set_map_random ()

  Set map_random.
- bool with_map_surjective () const

  Get map_surjective.
- void set_map_surjective ()

  Set map_surjective.
- bool with_mmas_strict () const

  Get mmas_strict.
- void set_mmas_strict ()

  Set mmas_strict.
- bool with_negation () const

  Get negation.
- void set_negation ()

  Set negation.
- bool with_parsed_modifier () const

  Get parsed_modifier.
- void set_parsed_modifier ()

  Set parsed_modifier.
- bool with_pn_allow_no_mutation () const

  Get pn_allow_no_mutation.
- void set_pn_allow_no_mutation ()

  Set pn_allow_no_mutation.
- bool with_print_defaults () const

  Get print_defaults.
- void set_print_defaults ()

  Set print_defaults.
- bool with_print_description () const

  Get print_description.
- void set_print_description ()

  Set print_description.
- bool with_print_header () const

Get print_header.
- void set_print_header ()

  Set print_header.
- bool with_print_results () const

  Get print_results.
- void set_print_results ()

  Set print_results.
- bool with_print_solution () const

  Get print_solution.
- void set_print_solution ()

  Set print_solution.
- bool with_prior_noise () const

  Get prior_noise.
- void set_prior_noise ()

  Set prior_noise.
- bool with_pv_log_entropy () const

  Get pv_log_entropy.
- void set_pv_log_entropy ()

  Set pv_log_entropy.
- bool with_pv_log_pv () const

  Get pv_log_pv.
- void set_pv_log_pv ()

  Set pv_log_pv.
- bool with_record_evaluation_time () const

  Get record_evaluation_time.
- void set_record_evaluation_time ()

  Set record_evaluation_time.
- bool with_restart () const

  Get restart.
- void set_restart ()

  Set restart.
- bool with_rls_strict () const

  Get rls_strict.
- void set_rls_strict ()

  Set rls_strict.
- bool with_rw_log_value () const

  Get rw_log_value.
- void set_rw_log_value ()

  Set rw_log_value.
- bool with_save_description () const

  Get save_description.
- void set_save_description ()

  Set save_description.
- bool with_save_results () const

  Get save_results.
- void set_save_results ()

  Set save_results.
- bool with_save_solution () const

  Get save_solution.
- void set_save_solution ()

  Set save_solution.

- bool with_stop_on_maximum () const

  Get stop_on_maximum.
- void set_stop_on_maximum ()

  Set stop_on_maximum.
- bool with_stop_on_target () const

  Get stop_on_target.
- void set_stop_on_target ()

  Set stop_on_target.
- Options (int argc, char ∗argv[ ])

  Constructor.
- int get_input_size () const

  Get input_size.
- void set_input_size (int x)

  Set input_size.
- bool set_input_size () const

  Get set-flag for input_size.
- int get_map () const

  Get map.
- void set_map (int x)

  Set map.
- bool set_map () const

  Get set-flag for map.
- int get_output_size () const

  Get output_size.
- void set_output_size (int x)

  Set output_size.
- bool set_output_size () const

  Get set-flag for output_size.
- std::string get_path () const

  Get path.
- void set_path (std::string x)

  Set path.
- bool set_path () const

  Get set-flag for path.
- int get_seed () const

  Get seed.
- void set_seed (int x)

  Set seed.
- bool set_seed () const

  Get set-flag for seed.
- int get_ts_length () const

  Get ts_length.
- void set_ts_length (int x)

  Set ts_length.
- bool set_ts_length () const

  Get set-flag for ts_length.
- int get_ts_sampling_mode () const

  Get ts_sampling_mode.
- void set_ts_sampling_mode (int x)

  Set ts_sampling_mode.
- bool set_ts_sampling_mode () const

Get set-flag for ts_sampling_mode.
- bool with_surjective () const

  Get surjective.
- void set_surjective ()

  Set surjective.

Private Member Functions

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

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

  Print version.
- void print_help (std::ostream &stream) const

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

  Print help message for section fp.
- 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_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.
- 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

  Name Version.
- int _bv_size

  Size of bit vectors.
- bool **_opt_bv_size**
- double _coupling_constant

Coupling constant.
- bool **_opt_coupling_constant**
- double _ep_upper_bound

  Upper bound of numbers.
- bool **_opt_ep_upper_bound**
- double _field_constant

  Field constant.
- bool **_opt_field_constant**
- int _function

  Type of function.
- bool **_opt_function**
- double _lin_distance

  Common distance of arithmetic progression.
- bool **_opt_lin_distance**
- int _lin_generator

  Type of LinearFunction generator.
- bool **_opt_lin_generator**
- double _lin_initial_weight

  Initial weight.
- bool **_opt_lin_initial_weight**
- double _lin_ratio

  Common ratio of geometric progression.
- bool **_opt_lin_ratio**
- int _ms_num_clauses

  Number of clauses.
- bool **_opt_ms_num_clauses**
- int _ms_num_literals_per_clause

  Number of literals per clause.
- bool **_opt_ms_num_literals_per_clause**
- int _nk_k

  Each bit is connected to k other bits.
- bool **_opt_nk_k**
- int _nn1_generator

  Type of NearestNeighborIsingModel1 generator.
- bool **_opt_nn1_generator**
- int _nn2_generator

  Type of NearestNeighborIsingModel2 generator.
- bool **_opt_nn2_generator**
- int _nn2_num_columns

  Number of columns.
- bool **_opt_nn2_num_columns**
- int _nn2_num_rows

  Number of rows.
- bool **_opt_nn2_num_rows**
- int _part_upper_bound

  Upper bound of numbers.
- bool **_opt_part_upper_bound**
- std::string _path

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

  Seed for the random number generator.

- bool **_opt_seed**
- double _stddev

  Standard deviation.
- bool **_opt_stddev**
- int _walsh2_generator

  Type of WalshExpansion2 generator.
- bool **_opt_walsh2_generator**
- double _walsh2_ising_alpha

  Dyson-Ising: exponential decay parameter for long range interactions.
- bool **_opt_walsh2_ising_alpha**
- int _walsh_num_features

  Number of features.
- bool **_opt_walsh_num_features**
- bool _ms_planted_solution

  Generate an instance with a planted solution.
- bool _periodic_boundary_conditions

  Periodic boundary conditions.
- int _algorithm

  Type of algorithm.
- bool **_opt_algorithm**
- int _bm_mc_reset_strategy

  Markov chain reset strategy.
- bool **_opt_bm_mc_reset_strategy**
- int _bm_num_gs_cycles

  Number of Gibbs sampler cycles per bit vector.
- bool **_opt_bm_num_gs_cycles**
- int _bm_num_gs_steps

  Number of Gibbs sampler steps per bit vector.
- bool **_opt_bm_num_gs_steps**
- int _bm_sampling

  Sampling mode for the Boltzmann machine.
- bool **_opt_bm_sampling**
- int _budget

  Number of allowed function evaluations (<= 0 means indefinite)
- bool **_opt_budget**
- std::string _description_path

  Path of the description file.
- bool **_opt_description_path**
- int _ea_lambda

  Offspring population size.
- bool **_opt_ea_lambda**
- int _ea_mu

  Parent population size.
- bool **_opt_ea_mu**
- std::string _expression

  Expression of the variable x.
- bool **_opt_expression**
- std::string _fn_name

  Name of the function in the dynamic library.
- bool **_opt_fn_name**
- int _fn_num_traps

  Number of traps.

- bool **_opt_fn_num_traps**
- int _fn_prefix_length

  Prefix length for long path.
- bool **_opt_fn_prefix_length**
- int _fn_threshold

  Threshold (in bits) for Jump, Four Peaks, and Six Peaks.
- bool **_opt_fn_threshold**
- std::string _fp_expression

  Expression to parse.
- bool **_opt_fp_expression**
- double _fp_lower_bound

  Lower bound.
- bool **_opt_fp_lower_bound**
- int _fp_num_bits

  Number of bits in the dyadic representation of a number.
- bool **_opt_fp_num_bits**
- double _fp_upper_bound

  Upper bound.
- bool **_opt_fp_upper_bound**
- double _ga_crossover_bias

  Crossover bias.
- bool **_opt_ga_crossover_bias**
- double _ga_crossover_probability

  Crossover probability.
- bool **_opt_ga_crossover_probability**
- int _ga_tournament_size

  Tournament size.
- bool **_opt_ga_tournament_size**
- int _hea_bit_herding

  Type of bit herding.
- bool **_opt_hea_bit_herding**
- int _hea_num_seq_updates

  Number of sequential updates per sample.
- bool **_opt_hea_num_seq_updates**
- int _hea_reset_period

  Reset period (<= 0 means no reset)
- bool **_opt_hea_reset_period**
- int _hea_sampling_method

  Sampling method for spin features.
- bool **_opt_hea_sampling_method**
- double _hea_weight

  Weight of second moments.
- bool **_opt_hea_weight**
- double _learning_rate

  Learning rate.
- bool **_opt_learning_rate**
- int _map

  Type of map.
- bool **_opt_map**
- int _map_input_size

  Input size of linear and affine maps.
- bool **_opt_map_input_size**

- std::string _map_path

    Path of a map file.
- bool **_opt_map_path**
- int _map_ts_length

    Transvection sequence length.
- bool **_opt_map_ts_length**
- int _map_ts_sampling_mode

    Transvection sequence sampling mode.
- bool **_opt_map_ts_sampling_mode**
- double _mutation_rate

    Mutation rate relative to bv_size.
- bool **_opt_mutation_rate**
- int _neighborhood

    Type of neighborhood.
- bool **_opt_neighborhood**
- int _neighborhood_iterator

    Type of neighborhood iterator.
- bool **_opt_neighborhood_iterator**
- double _noise_stddev

    Noise standard deviation.
- bool **_opt_noise_stddev**
- int _num_iterations

    Number of iterations (<= 0 means indefinite)
- bool **_opt_num_iterations**
- int _num_threads

    Number of threads.
- bool **_opt_num_threads**
- double _pn_mutation_rate

    Mutation rate relative to bv_size.
- bool **_opt_pn_mutation_rate**
- int _pn_neighborhood

    Type of neighborhood.
- bool **_opt_pn_neighborhood**
- int _pn_radius

    Radius of Hamming ball or sphere.
- bool **_opt_pn_radius**
- int _population_size

    Population size.
- bool **_opt_population_size**
- int _pv_log_num_components

    Number of probability vector components to log.
- bool **_opt_pv_log_num_components**
- int _radius

    Radius of Hamming ball or sphere.
- bool **_opt_radius**
- std::string _results_path

    Path of the results file.
- bool **_opt_results_path**
- int _rls_patience

    Number of consecutive rejected moves before throwing LocalMaximumReached (<= 0 means infinite)
- bool **_opt_rls_patience**
- double _sa_beta_ratio

Ratio for beta or inverse temperature.

- bool **_opt_sa_beta_ratio**
- double _sa_initial_acceptance_probability

  Initial acceptance probability.
- bool **_opt_sa_initial_acceptance_probability**
- int _sa_num_transitions

  Number of accepted transitions before annealing.
- bool **_opt_sa_num_transitions**
- int _sa_num_trials

  Number of trials to estimate initial inverse temperature.
- bool **_opt_sa_num_trials**
- unsigned _seed

  Seed for the random number generator.
- int _selection_size

  Selection size (number of selected individuals)
- bool **_opt_selection_size**
- std::string _solution_path

  Path of the solution file.
- bool **_opt_solution_path**
- double _target

  Target.
- bool **_opt_target**
- bool _additive_gaussian_noise

  Additive Gaussian noise.
- bool _allow_no_mutation

  Allow no mutation with standard bit mutation.
- bool _bm_log_norm_infinite

  Log infinite norm of the parameters.
- bool _bm_log_norm_l1

  Log L1 norm of the parameters.
- bool _bm_negative_positive_selection

  Negative and positive selection.
- bool _cache

  Cache function evaluations.
- bool _cache_budget

  Set cache on budget.
- bool _concrete_solution

  At the end, print or save the solution in the domain of the concrete function.
- bool _fn_display

  Display the function and exit.
- bool _fn_get_bv_size

  Print the size of bit vectors.
- bool _fn_get_maximum

  If the maximum is known then print it and exit with status 0 else exit with status 1.
- bool _fn_has_known_maximum

  Does the function have a known maximum?
- bool _fn_provides_incremental_evaluation

  Does the function provide incremental evaluation?
- bool _fn_walsh_transform

  Compute the Walsh transform of the function.
- bool _hea_bound_moment

Bound moment after update.
- bool _hea_log_delta

  Log norm 2 of delta (in moment space)
- bool _hea_log_dtu

  Log distance to uniform.
- bool _hea_log_error

  Log error (moment discrepancy)
- bool _hea_log_moment_matrix

  Log moment matrix.
- bool _hea_log_selection

  Log the distance between the target and the selection moment.
- bool _hea_randomize_bit_order

  Randomize bit order.
- bool _incremental_evaluation

  Incremental evaluation.
- bool _load_solution

  Load a solution from a file.
- bool _log_improvement

  Log improvement.
- bool _map_display

  Display the map and exit.
- bool _map_random

  Sample a random map.
- bool _map_surjective

  Ensure that the sampled linear or affine map is surjective.
- bool _mmas_strict

  Strict (>) max-min ant system.
- bool _negation

  Negation (hence minimization) of the function.
- bool _parsed_modifier

  Parsed modifier.
- bool _pn_allow_no_mutation

  Allow no mutation with standard bit mutation.
- bool _print_defaults

  Print the default parameters and exit.
- bool _print_description

  Print a description of the solution.
- bool _print_header

  At the beginning, print the header.
- bool _print_results

  Print results.
- bool _print_solution

  Print the solution.
- bool _prior_noise

  Prior noise.
- bool _pv_log_entropy

  Log entropy of probability vector.
- bool _pv_log_pv

  Log probability vector.
- bool _record_evaluation_time

  Record evaluation time.

- bool _restart

  Restart any algorithm an indefinite number of times.
- bool _rls_strict

  Strict (>) random local search.
- bool _rw_log_value

  Log bit vector value during random walk.
- bool _save_description

  At the end, save a description of the solution in a file.
- bool _save_results

  At the end, save results in a file.
- bool _save_solution

  At the end, save the solution in a file.
- bool _stop_on_maximum

  Stop on maximum.
- bool _stop_on_target

  Stop on target.
- int _input_size

  Input bit vector size.
- bool **_opt_input_size**
- int _output_size

  Output bit vector size.
- bool **_opt_output_size**
- int _ts_length

  Transvection sequence length.
- bool **_opt_ts_length**
- int _ts_sampling_mode

  Transvection sequence sampling mode.
- bool **_opt_ts_sampling_mode**
- bool _surjective

  Ensure that the sampled linear or affine map is surjective.

Friends

- std::ostream & operator<< (std::ostream &, const Options &)

  Print a header containing the parameter values.
- std::ostream & operator<< (std::ostream &, const Options &)

  Print a header containing the parameter values.
- std::ostream & operator<< (std::ostream &, const Options &)

  Print a header containing the parameter values.

5.87.1 Detailed Description

Command line options.

Definition at line 8 of file ffgen-options.hh.

5.87.2 Member Data Documentation

5.87.2.1 _path

```
std::string _path [private]
```

Path (relative or absolute) of a function file.

Path (relative or absolute) of a map file.

Path of a function file.

Definition at line 85 of file ffgen-options.hh.

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

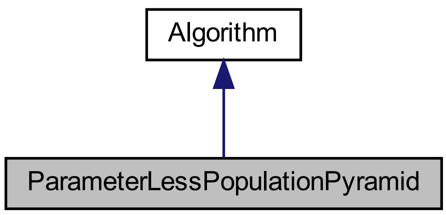
- app/ffgen-options.hh
- app/hnco-options.hh
- app/mapgen-options.hh
- app/ffgen-options.cc
- app/hnco-options.cc
- app/mapgen-options.cc

5.88 ParameterLessPopulationPyramid Class Reference

Parameter-less Population Pyramid.

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

Inheritance diagram for ParameterLessPopulationPyramid:



Public Member Functions

- ParameterLessPopulationPyramid (int n)

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

  Maximize.
- void finalize ()

  Finalize.

Private Attributes

- std::unique_ptr< Implementation > _pimpl

  Pointer to implementation.

Additional Inherited Members

5.88.1 Detailed Description

Parameter-less Population Pyramid.

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

Author: Brian W. Goldman

Reference:

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

Integrated into HNCO by Arnaud Berny

Definition at line 53 of file p3.hh.

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

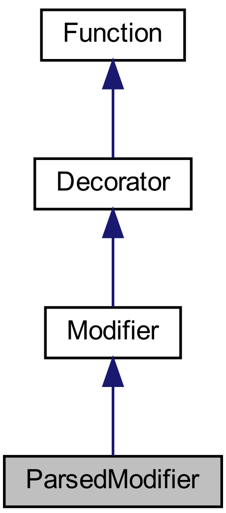
- lib/hnco/algorithms/fast-efficient-p3/p3.hh
- lib/hnco/algorithms/fast-efficient-p3/p3.cc

5.89 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 ()

  Get bit vector size.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.

Private Attributes

- FunctionParser _fparser

  Function parser.
- double _values [1]

  Array of values.

Additional Inherited Members

5.89.1 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.

5.89.2 Constructor & Destructor Documentation

5.89.2.1 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:

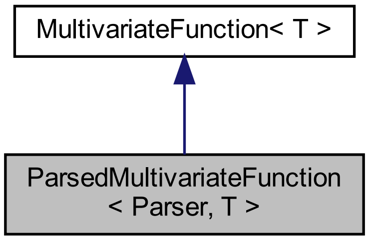
- lib/hnco/functions/modifiers/parsed-modifier.hh
- lib/hnco/functions/modifiers/parsed-modifier.cc

5.90 ParsedMultivariateFunction< Parser, T > Class Template Reference

Parsed multivariate function.

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

Inheritance diagram for ParsedMultivariateFunction< Parser, T >:



Public Member Functions

- ParsedMultivariateFunction (std::string expression)

  Constructor.
- int get_num_variables ()

  Get the number of variables.
- T evaluate (const std::vector< T > &x)

  Evaluate.

Private Attributes

- Parser _fparser

  Function parser.
- int _num_variables = 0

  Number of variables.

Additional Inherited Members

5.90.1 Detailed Description

template<class Parser, class T>
class hnco::function::representation::ParsedMultivariateFunction< Parser, T >

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 66 of file multivariate-function.hh.

5.90.2 Constructor & Destructor Documentation

5.90.2.1 ParsedMultivariateFunction()

```
ParsedMultivariateFunction (
            std::string expression )  [inline]
```

Constructor.

Parameters

expression	Expression to parse

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

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

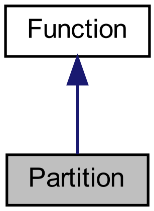
- lib/hnco/functions/representations/multivariate-function.hh

5.91 Partition Class Reference

Partition.

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

Inheritance diagram for Partition:



Public Member Functions

- Partition ()

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.

Instance generators

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

  Instance generator.
- void random (int n, int upper_bound)

  Random instance.

Display

- void display (std::ostream &stream)

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

  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.

Friends

- class **boost::serialization::access**

5.91.1 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 53 of file partition.hh.

5.91.2 Member Function Documentation

5.91.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Number generator

Definition at line 86 of file partition.hh.

5.91.2.2 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 102 of file partition.hh.

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

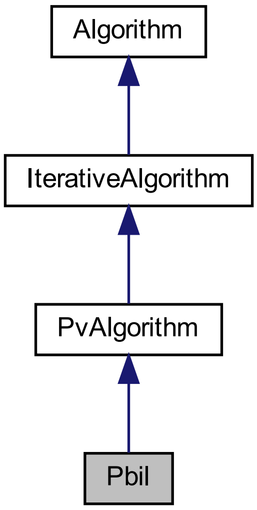
- lib/hnco/functions/partition.hh
- lib/hnco/functions/partition.cc

5.92 Pbil Class Reference

Population-based incremental learning.

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

Inheritance diagram for Pbil:



Public Member Functions

- Pbil (int n, int population_size)
  Constructor.

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.

5.92.1 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:

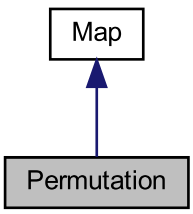
- lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

5.93 Permutation Class Reference

Permutation.

`#include <hnco/map.hh>`

Inheritance diagram for Permutation:



Public Member Functions

- void random (int n)

    Random instance.
- void map (const bit_vector_t &input, bit_vector_t &output)

    Map
- int get_input_size ()

    Get input size.
- int get_output_size ()

    Get output size.
- bool is_surjective ()

    Check for surjective map.

Private Member Functions

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

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

    Load.

Private Attributes

- permutation_t _permutation

    Permutation.

Friends

- class **boost::serialization::access**

5.93.1 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 139 of file map.hh.

5.93.2 Member Function Documentation

5.93.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 190 of file map.hh.

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

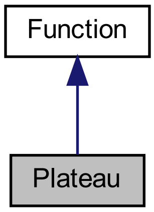
- lib/hnco/map.hh
- lib/hnco/map.cc

5.94 Plateau Class Reference

Plateau.

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

Inheritance diagram for Plateau:



Public Member Functions

- Plateau (int bv_size)

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- bool has_known_maximum ()

  Check for a known maximum.
- double get_maximum ()

  Get the global maximum.

Private Attributes

- int _bv_size
  Bit vector size.

5.94.1 Detailed Description

Plateau.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 244 of file theory.hh.

5.94.2 Member Function Documentation

5.94.2.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size + 2

Reimplemented from Function.

Definition at line 268 of file theory.hh.

5.94.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 264 of file theory.hh.

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

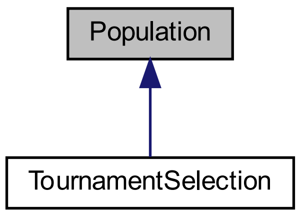
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.95 Population Class Reference

Population

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

Inheritance diagram for Population:



Public Member Functions

- Population (int population_size, int n)

  Constructor.
- int size () const

  Size.
- void random ()

  Initialize the population with random bit vectors.

Get bit vectors for non const populations

- bit_vector_t & get_bv (int i)

  Get a bit vector.
- bit_vector_t & get_best_bv ()

  Get best bit vector.
- bit_vector_t & get_best_bv (int i)

  Get best bit vector.
- bit_vector_t & get_worst_bv (int i)

  Get worst bit vector.

Get bit vectors for const populations

- const bit_vector_t & get_bv (int i) const

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

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

  Get best bit vector.
- const bit_vector_t & get_worst_bv (int i) const

  Get worst bit vector.

Get sorted values

- double get_best_value (int i) const

  Get best value.
- double get_best_value () const

  Get best value.

Evaluation and sorting

- void evaluate (function::Function ∗function)

  Evaluate the population.
- void evaluate_in_parallel (const std::vector< function::Function ∗ > &functions)

  Evaluate the population in parallel.
- void sort ()

  Sort the lookup table.
- void partial_sort (int selection_size)

  Partially sort the lookup table.
- void shuffle ()

  Shuffle the lookup table.

Selection

- void plus_selection (const Population &offsprings)

  Plus selection.
- void plus_selection (Population &offsprings)

  Plus selection.
- void comma_selection (const Population &offsprings)

  Comma selection.
- void comma_selection (Population &offsprings)

  Comma selection.

Protected Types

- typedef std::pair< int, double > index_value_t

  Index-value type.

Protected Attributes

- std::vector< bit_vector_t > _bvs

  Bit vectors.
- std::vector< index_value_t > _lookup

  Lookup table.
- std::function< bool(const index_value_t &, const index_value_t &)> _compare_index_value

  Binary operator for comparing index-value pairs.

5.95.1 Detailed Description

Population

Definition at line 36 of file population.hh.

5.95.2 Member Function Documentation

5.95.2.1 comma_selection() [1/2]

```
void comma_selection (
            const Population & offsprings )
```

Comma selection.

Implemented with a copy.

Precondition

Offspring population must be partially sorted.

Warning

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

Definition at line 117 of file population.cc.

5.95.2.2 comma_selection() [2/2]

```
void comma_selection (
            Population & offsprings )
```

Comma selection.

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

Precondition

Offspring population must be partially sorted.

Warning

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

Modifies its argument.

Definition at line 131 of file population.cc.

5.95.2.3 get_best_bv() [1/4]

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

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 83 of file population.hh.

5.95.2.4 get_best_bv() [2/4]

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

Get best bit vector.

Precondition

The population must be sorted.

Definition at line 115 of file population.hh.

5.95.2.5 get_best_bv() [3/4]

```
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 91 of file population.hh.

5.95.2.6 get_best_bv() [4/4]

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

Get best bit vector.

Parameters

i	Index in the sorted population

Precondition

The population must be sorted.

Definition at line 123 of file population.hh.

5.95.2.7 get_best_value() [1/2]

```
double get_best_value ( ) const [inline]
```

Get best value.

Precondition

The population must be sorted.

Definition at line 152 of file population.hh.

5.95.2.8 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 146 of file population.hh.

5.95.2.9 get_worst_bv() [1/2]

```
bit_vector_t& get_worst_bv (
              int i ) [inline]
```

Get worst bit vector.

Parameters

i	Index in the sorted population

Precondition

The population must be sorted.

Definition at line 99 of file population.hh.

5.95.2.10 get_worst_bv() [2/2]

```
const bit_vector_t& get_worst_bv (
              int i ) const [inline]
```

Get worst bit vector.

Parameters

i	Index in the sorted population

Precondition

The population must be sorted.

Definition at line 131 of file population.hh.

5.95.2.11 plus_selection() [1/2]

```
void plus_selection (
              const Population & offsprings )
```

Plus selection.

Implemented with a copy.

Precondition

Both populations must be completely sorted.

Warning

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

Definition at line 79 of file population.cc.

5.95.2.12 plus_selection() [2/2]

```
void plus_selection (
            Population & offsprings )
```

Plus selection.

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

Precondition

Both populations must be completely sorted.

Warning

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

Modifies its argument.

Definition at line 98 of file population.cc.

5.95.3 Member Data Documentation

5.95.3.1 _compare_index_value

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

Initial value:

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

Binary operator for comparing index-value pairs.

Definition at line 55 of file population.hh.

5.95.3.2 _lookup

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

Lookup table.

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

Definition at line 52 of file population.hh.

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

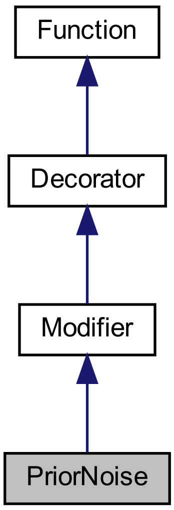
- lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

5.96 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 ()
  Get bit vector size.
- double get_maximum ()
  Get the global maximum.
- bool has_known_maximum ()
  Check for a known maximum.
- bool provides_incremental_evaluation ()
  Check whether the function provides incremental evaluation.

Evaluation

- double evaluate (const bit_vector_t &)
  Evaluate a bit vector.

Private Attributes

- neighborhood::Neighborhood ∗ _neighborhood
  Neighborhood.
- bit_vector_t _noisy_bv
  Noisy bit vector.

Additional Inherited Members

5.96.1 Detailed Description

Prior noise.

Definition at line 37 of file prior-noise.hh.

5.96.2 Member Function Documentation

5.96.2.1 get_maximum()

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

Get the global maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 69 of file prior-noise.hh.

5.96.2.2 has_known_maximum()

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

Check for a known maximum.

Delegation is questionable here.

Reimplemented from Function.

Definition at line 75 of file prior-noise.hh.

5.96.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from Function.

Definition at line 79 of file prior-noise.hh.

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

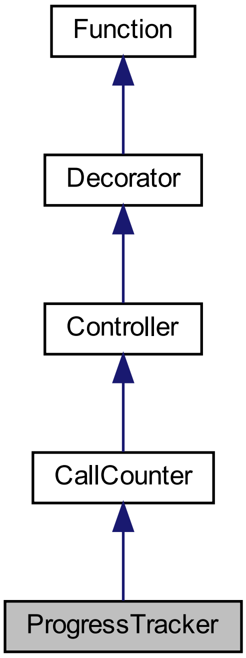
- lib/hnco/functions/modifiers/prior-noise.hh
- lib/hnco/functions/modifiers/prior-noise.cc

5.97 ProgressTracker Class Reference

ProgressTracker.

```
#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 &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &x, double value)
  Update after a safe evaluation.

Get information

- const Event & get_last_improvement ()

Get the last improvement.
- double get_evaluation_time ()

  Get evaluation time.

Setters

- void set_log_improvement (bool x)

  Log improvement.
- void set_record_evaluation_time (bool b)

  Record evaluation time.

Protected Member Functions

- void update_last_improvement (double value)

  Update last improvement.

Protected Attributes

- Event _last_improvement

  Last improvement.
- StopWatch _stop_watch

  Stop watch.

Parameters

- bool _log_improvement = false

  Log improvement.
- bool _record_evaluation_time = false

  Record evaluation time.

5.97.1 Detailed Description

ProgressTracker.

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

Definition at line 218 of file controller.hh.

5.97.2 Member Function Documentation

5.97.2.1 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 292 of file controller.hh.

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

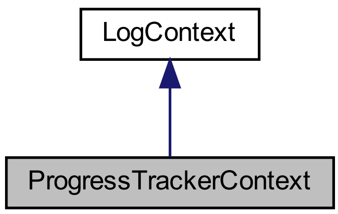
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.98 ProgressTrackerContext Class Reference

Log context for ProgressTracker.

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

Inheritance diagram for ProgressTrackerContext:



Public Member Functions

- ProgressTrackerContext (hnco::function::controller::ProgressTracker ∗pt)

  Constructor.
- std::string to_string ()

  Get context.

Private Attributes

- hnco::function::controller::ProgressTracker ∗ _pt

  Progress tracker.

5.98.1 Detailed Description

Log context for ProgressTracker.

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

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

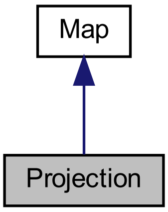
- lib/hnco/logging/log-context.hh

5.99 Projection Class Reference

Projection.

`#include <hnco/map.hh>`

Inheritance diagram for Projection:



Public Member Functions

- Projection (const std::vector< int > &bit_positions, int input_size)
  Constructor.
- void map (const bit_vector_t &input, bit_vector_t &output)
  Map
- int get_input_size ()
  Get input size.
- int get_output_size ()
  Get output size.
- bool is_surjective ()
  Check for surjective map.

Private Attributes

- std::vector< int > _bit_positions
  Bit positions.
- int _input_size
  Input size.

5.99.1 Detailed Description

Projection.

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

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

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

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 459 of file map.hh.

5.99.2 Constructor & Destructor Documentation

5.99.2.1 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 164 of file map.cc.

5.99.3 Member Function Documentation

5.99.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 497 of file map.hh.

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

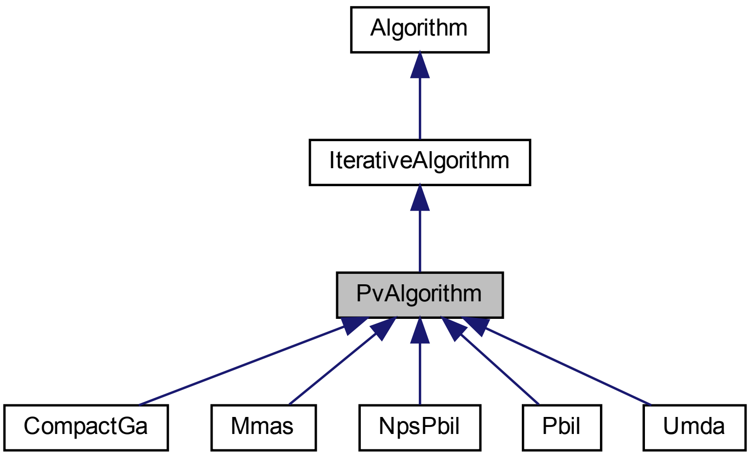
- lib/hnco/map.hh
- lib/hnco/map.cc

5.100 PvAlgorithm Class Reference

Probability vector algorithm.

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

Inheritance diagram for PvAlgorithm:



Public Member Functions

- PvAlgorithm (int n)

  Constructor.

Setters for logging

- void set_log_entropy (bool x)

  Log entropy.
- void set_log_num_components (int x)

  Set the number of probability vector components to log.
- void set_log_pv (bool x)

  Log probability vector.

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.

5.100.1 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:

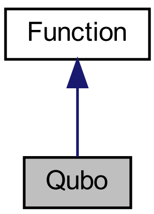
- lib/hnco/algorithms/pv/pv-algorithm.hh
- lib/hnco/algorithms/pv/pv-algorithm.cc

5.101 Qubo Class Reference

Quadratic unconstrained binary optimization.

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

Inheritance diagram for Qubo:



Public Member Functions

- Qubo ()
  Constructor.
- void load (std::istream &stream)
  Load an instance.
- int get_bv_size ()
  Get bit vector size.
- double evaluate (const bit_vector_t &)
  Evaluate a bit vector.

Private Attributes

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

5.101.1 Detailed Description

Quadratic unconstrained binary optimization.

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

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

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

There are some differences between WalshExpansion2 and Qubo:

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

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

References:

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

https://github.com/dwavesystems/qbsolv

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

Definition at line 74 of file qubo.hh.

5.101.2 Member Function Documentation

5.101.2.1 load()

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

Load an instance.

Exceptions

Error	

Definition at line 35 of file qubo.cc.

5.101.3 Member Data Documentation

5.101.3.1 _q

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

Matrix.

n x n upper triangular matrix.

Definition at line 83 of file qubo.hh.

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

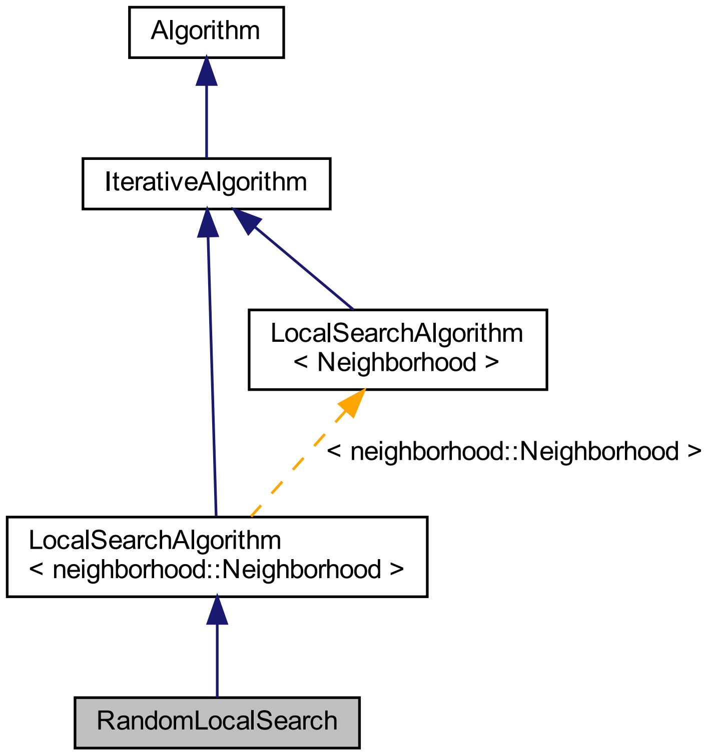
- lib/hnco/functions/qubo.hh
- lib/hnco/functions/qubo.cc

5.102 RandomLocalSearch Class Reference

Random local search.

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

Inheritance diagram for RandomLocalSearch:



Public Member Functions

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

    Constructor.
- void finalize ()

    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.

5.102.1 Detailed Description

Random local search.

Definition at line 36 of file random-local-search.hh.

5.102.2 Member Function Documentation

5.102.2.1 set_patience()

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

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximumReached exception

Parameters

x	Patience

If x <= 0 then patience is considered infinite, meaning that the algorithm will never throw any LocalMaximum↩

Reached exception.

Definition at line 103 of file random-local-search.hh.

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

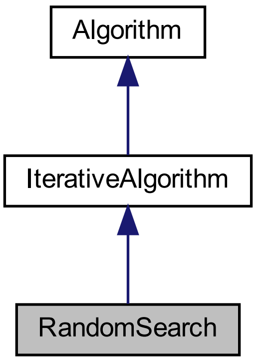
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

5.103 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.

5.103.1 Detailed Description

Random search.

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

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

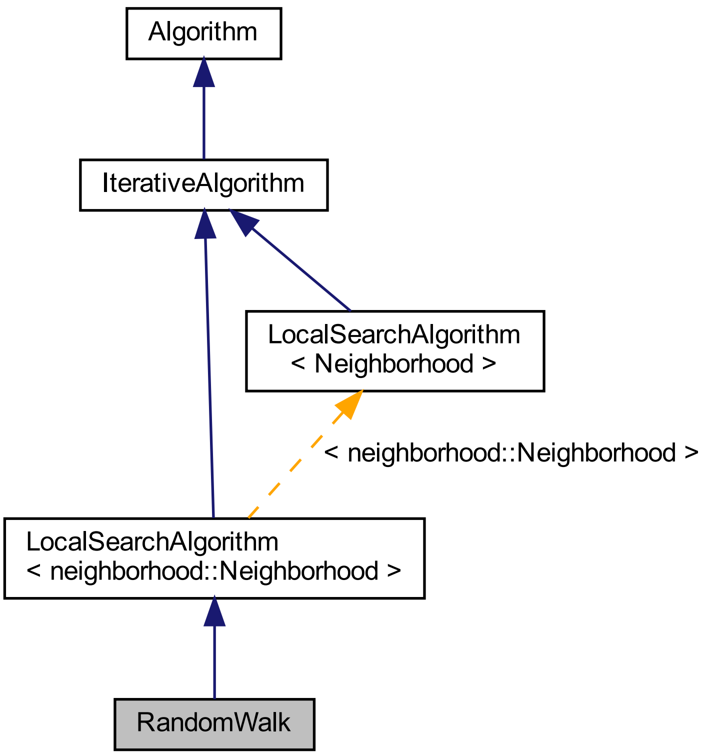
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

5.104 RandomWalk Class Reference

Random walk.

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

Inheritance diagram for RandomWalk:



Public Member Functions

- RandomWalk (int n, neighborhood::Neighborhood *neighborhood)
  Constructor.

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.

5.104.1 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:

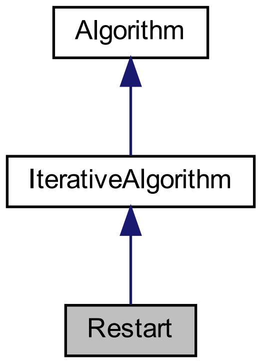
- lib/hnco/algorithms/ls/random-walk.hh
- lib/hnco/algorithms/ls/random-walk.cc

5.105 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.

5.105.1 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:

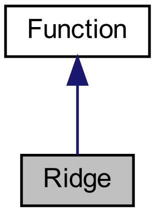
- lib/hnco/algorithms/decorators/restart.hh
- lib/hnco/algorithms/decorators/restart.cc

5.106 Ridge Class Reference

Ridge.

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

Inheritance diagram for Ridge:



Public Member Functions

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

Private Attributes

- int _bv_size
  Bit vector size.

5.106.1 Detailed Description

Ridge.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 208 of file theory.hh.

5.106.2 Member Function Documentation

5.106.2.1 get_maximum()

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

Get the global maximum.

Returns

2 * _bv_size

Reimplemented from Function.

Definition at line 232 of file theory.hh.

5.106.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 228 of file theory.hh.

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

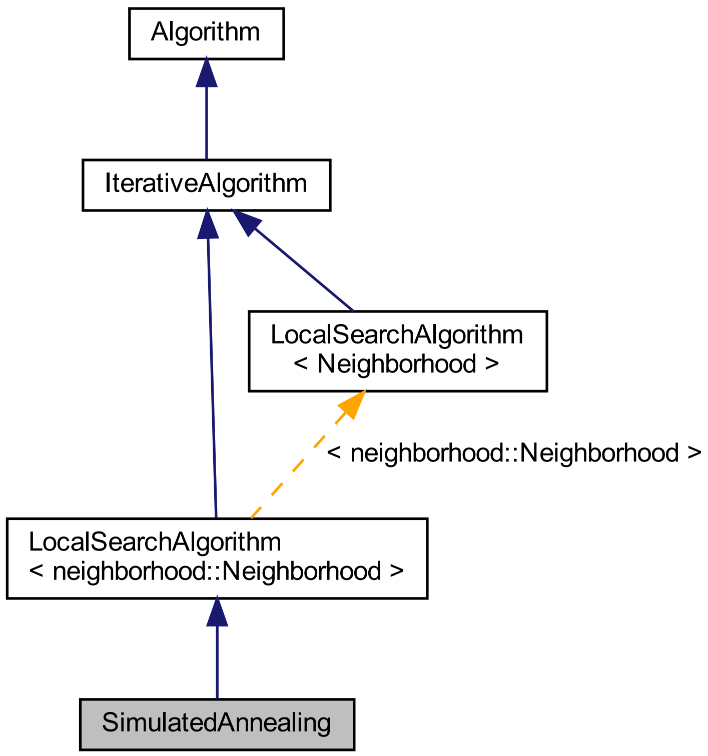
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

5.107 SimulatedAnnealing Class Reference

Simulated annealing.

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

Inheritance diagram for SimulatedAnnealing:



Public Member Functions

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

  Constructor.

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.

5.107.1 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.

5.107.2 Member Function Documentation

5.107.2.1 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:

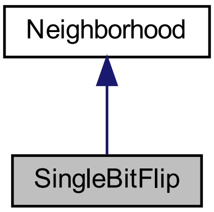
- lib/hnco/algorithms/ls/simulated-annealing.hh
- lib/hnco/algorithms/ls/simulated-annealing.cc

5.108 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

5.108.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

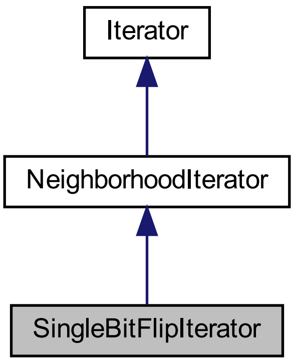
- lib/hnco/neighborhoods/neighborhood.hh

5.109 SingleBitFlipIterator Class Reference

Single bit flip neighborhood iterator.

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

Inheritance diagram for SingleBitFlipIterator:



Public Member Functions

- SingleBitFlipIterator (int n)

  Constructor.
- bool has_next ()

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

  Next bit vector.

Private Attributes

- size_t _index
  Index of the last flipped bit.

Additional Inherited Members

5.109.1 Detailed Description

Single bit flip neighborhood iterator.

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

5.109.2 Constructor & Destructor Documentation

5.109.2.1 SingleBitFlipIterator()

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

Constructor.

Parameters

n	Size of bit vectors

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

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

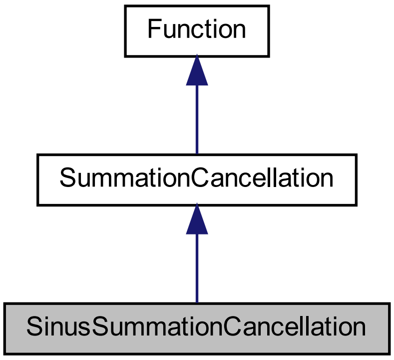
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

5.110 SinusSummationCancellation Class Reference

Summation cancellation with sinus.

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

Inheritance diagram for SinusSummationCancellation:



Public Member Functions

- SinusSummationCancellation (int n)

  Constructor.
- double evaluate (const bit_vector_t &x)

  Evaluate a bit vector.

Additional Inherited Members

5.110.1 Detailed Description

Summation cancellation with sinus.

Reference:

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

Definition at line 104 of file cancellation.hh.

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

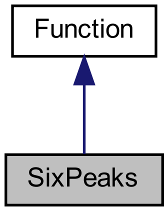
- lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.111 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



Public Member Functions

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

Private Attributes

- int _bv_size
  Bit vector size.
- int _threshold
  Threshold.
- int _maximum
  Maximum.

5.111.1 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.

5.111.2 Member Function Documentation

5.111.2.1 get_maximum()

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

Get the global maximum.

Returns

2 $*$ _bv_size - _threshold - 1

Reimplemented from Function.

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

5.111.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

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

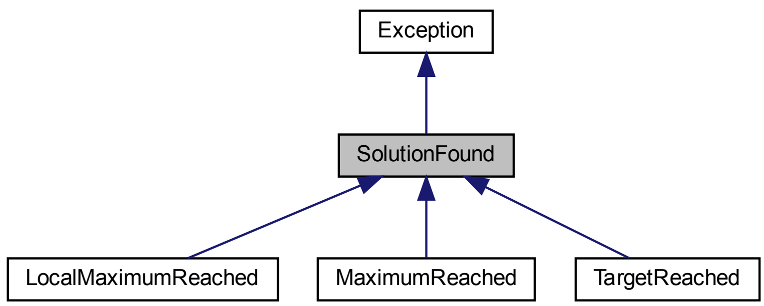
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

5.112 SolutionFound Class Reference

Solution found.

`#include <hnco/exception.hh>`

Inheritance diagram for SolutionFound:



Public Member Functions

- SolutionFound (const algorithm::solution_t &solution)
  Constructor.
- const algorithm::solution_t & get_solution () const
  Get solution.

Protected Attributes

- algorithm::solution_t _solution
  Solution.

5.112.1 Detailed Description

Solution found.

Definition at line 41 of file exception.hh.

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

- lib/hnco/exception.hh

5.113 SpinHerding Class Reference

Herding with spin variables.

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

Public Types

- enum { SAMPLE_GREEDY, SAMPLE_RLS, SAMPLE_DLS, **LAST_SAMPLE** }

Public Member Functions

- SpinHerding (int n)
  Constructor.
- void init ()
  Initialization.
- void sample (const SpinMoment &target, bit_vector_t &x)
  Sample a bit vector.
- double error (const SpinMoment &target)
  Compute the error.

Getters

- const SpinMoment & get_delta ()
  Get delta.

Setters

- void set_randomize_bit_order (bool x)
  Randomize bit order.
- void set_sampling_method (int x)
  Set the sampling method.
- void set_num_seq_updates (int x)
  Set the number of sequential updates per sample.
- void set_weight (double x)
  Set the weight of second order moments.

Protected Member Functions

- void compute_delta (const SpinMoment &target)
  Compute delta.
- void sample_greedy (bit_vector_t &x)
  Sample by means of a greedy algorithm.
- double q_derivative (const bit_vector_t &x, int i)
  Derivative of q.
- double q_variation (const bit_vector_t &x, int i)
  Variation of q.
- void sample_rls (bit_vector_t &x)
  Sample by means of random local search.
- void sample_dls (bit_vector_t &x)
  Sample by means of deterministic local search.

Protected Attributes

- SpinMoment _delta
  Delta moment.
- SpinMoment _count
  Counter moment.
- permutation_t _permutation
  Permutation.
- std::uniform_int_distribution< int > _choose_bit
  Choose bit.
- int _time
  Time.

Parameters

- bool _randomize_bit_order = false
  Randomize bit order.
- int _sampling_method = SAMPLE_GREEDY
  Sampling method.
- int _num_seq_updates
  Number of sequential updates per sample.
- double _weight = 1
  Weight of second order moments.

5.113.1 Detailed Description

Herding with spin variables.

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

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

5.113.2 Member Enumeration Documentation

5.113.2.1 anonymous enum

```
anonymous enum
```

Enumerator

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

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

5.113.3 Constructor & Destructor Documentation

5.113.3.1 SpinHerding()

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

Constructor.

Parameters

n	Size of bit vectors

_num_seq_updates is initialized to n.

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

5.113.4 Member Function Documentation

5.113.4.1 q_variation()

```
double q_variation (
            const bit_vector_t & x,
            int i )  [protected]
```

Variation of q.

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

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

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

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

5.114 SpinMoment Struct Reference

Moment for spin variables.

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

Public Member Functions

- SpinMoment (int n)

  Constructor.
- void uniform ()

  Set the moment to that of the uniform distribution.
- void init ()

  Initialize accumulators.
- void add (const bit_vector_t &x)

  Update accumulators.
- void average (int count)

  Compute average.
- void update (const SpinMoment &p, double rate)

  Update moment.
- void bound (double margin)

  Bound moment.
- double distance (const SpinMoment &p) const

  Distance.
- double norm_2 () const

  Compute the norm 2.
- double diameter () const

  Compute the diameter.
- size_t size () const

  Size.
- void display (std::ostream &stream)

  Display.

Public Attributes

- std::vector< double > _first

  First moment.
- std::vector< std::vector< double > > _second

  Second moment.
- double _weight = 1

  Weight of second order moments.

5.114.1 Detailed Description

Moment for spin variables.

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

5.114.2 Member Data Documentation

5.114.2.1 _second

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

Second moment.

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

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

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

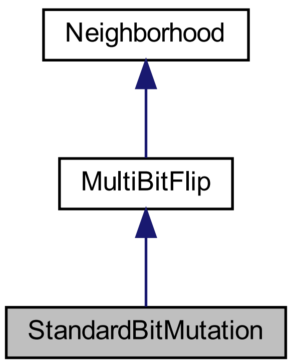
- lib/hnco/algorithms/hea/spin-moment.hh
- lib/hnco/algorithms/hea/spin-moment.cc

5.115 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.
- void set_mutation_rate (double p)

    Set mutation rate.

Setters

- 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

5.115.1 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 220 of file neighborhood.hh.

5.115.2 Constructor & Destructor Documentation

5.115.2.1 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 255 of file neighborhood.hh.

5.115.2.2 StandardBitMutation() [2/2]

```
StandardBitMutation (
            int n,
            double p ) [inline]
```

Constructor.

Parameters

n	Size of bit vectors
p	Bernoulli probability

Definition at line 265 of file neighborhood.hh.

5.115.3 Member Function Documentation

5.115.3.1 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 276 of file neighborhood.hh.

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

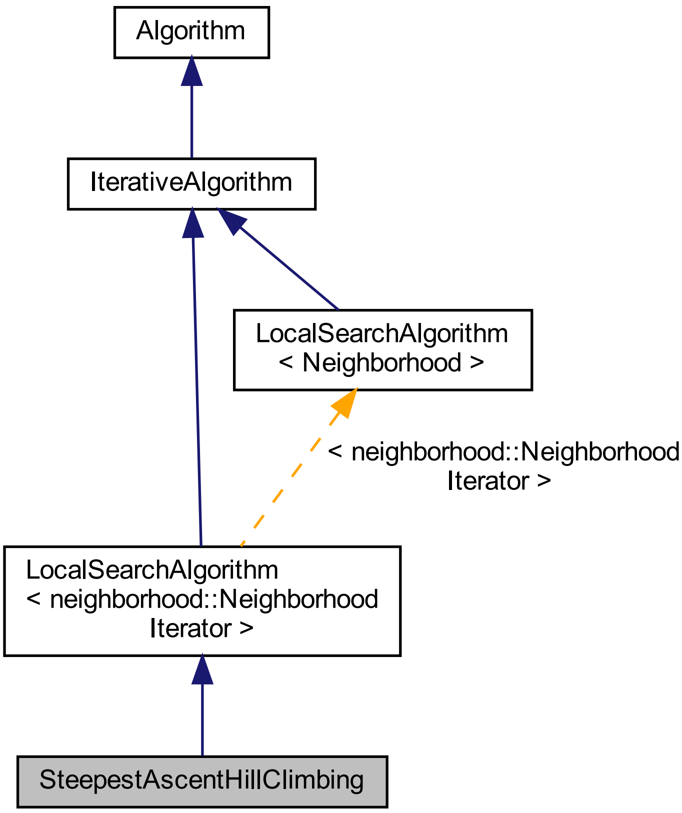
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

5.116 SteepestAscentHillClimbing Class Reference

Steepest ascent hill climbing.

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

Inheritance diagram for SteepestAscentHillClimbing:



Public Member Functions

- SteepestAscentHillClimbing (int n, neighborhood::NeighborhoodIterator ∗neighborhood)

  Constructor.

Protected Member Functions

- void iterate () override

  Single iteration.

Protected Attributes

- std::vector< bit_vector_t > _candidates

    Potential candidate.

5.116.1 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:

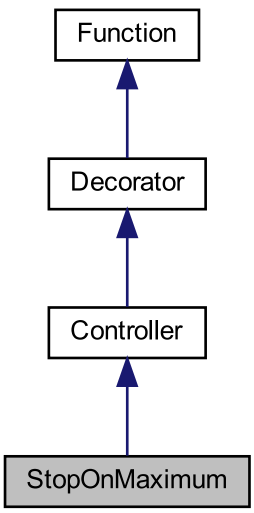
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

5.117 StopOnMaximum Class Reference

Stop on maximum.

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

Inheritance diagram for StopOnMaximum:



Public Member Functions

- StopOnMaximum (Function *function)

  Constructor.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &x, double value)

  Update after a safe evaluation.

Additional Inherited Members

5.117.1 Detailed Description

Stop on maximum.

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

Warning

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

Definition at line 92 of file controller.hh.

5.117.2 Constructor & Destructor Documentation

5.117.2.1 StopOnMaximum()

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

Constructor.

Parameters

function	Decorated function

Precondition

function->has_known_maximum()

Definition at line 100 of file controller.hh.

5.117.3 Member Function Documentation

5.117.3.1 evaluate()

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

Evaluate a bit vector.

Exceptions

MaximumReached	

Implements Function.

Definition at line 33 of file controller.cc.

5.117.3.2 evaluate_incrementally()

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

Incrementally evaluate a bit vector.

Exceptions

MaximumReached	

Reimplemented from Function.

Definition at line 45 of file controller.cc.

5.117.3.3 update()

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

Update after a safe evaluation.

Exceptions

MaximumReached	

Reimplemented from Function.

Definition at line 57 of file controller.cc.

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

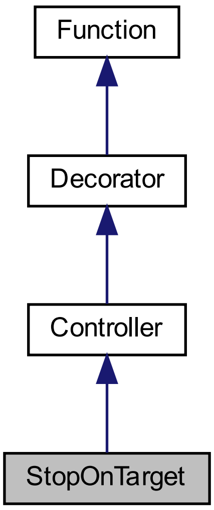
- lib/hnco/functions/controllers/controller.hh
- lib/hnco/functions/controllers/controller.cc

5.118 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.

Evaluation

- double evaluate (const bit_vector_t &)

  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double value, const hnco::sparse_bit_vector_t &flipped_bits)

  Incrementally evaluate a bit vector.
- void update (const bit_vector_t &x, double value)

  Update after a safe evaluation.

Private Attributes

- double _target

  Target.

Additional Inherited Members

5.118.1 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 136 of file controller.hh.

5.118.2 Constructor & Destructor Documentation

5.118.2.1 StopOnTarget()

```
StopOnTarget (
            Function * function,
            double target )  [inline]
```

Constructor.

Parameters

function	Decorated function
target	Target

Definition at line 149 of file controller.hh.

5.118.3 Member Function Documentation

5.118.3.1 evaluate()

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

Evaluate a bit vector.

Exceptions

TargetReached	

Implements Function.

Definition at line 68 of file controller.cc.

5.118.3.2 evaluate_incrementally()

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

Incrementally evaluate a bit vector.

Exceptions

TargetReached	

Reimplemented from Function.

Definition at line 78 of file controller.cc.

5.118.3.3 update()

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

Update after a safe evaluation.

Exceptions

TargetReached	

Reimplemented from Function.

Definition at line 88 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

5.119 StopWatch Class Reference

Stop watch.

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

Public Member Functions

- void start ()
  Start.
- void stop ()
  Stop.
- double get_total_time ()
  Get total time.

Private Attributes

- double _total_time = 0
  Total time.
- clock_t _start
  Start time.

5.119.1 Detailed Description

Stop watch.

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

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

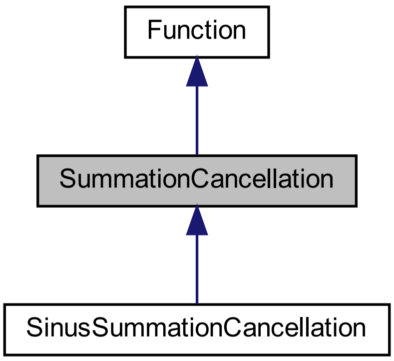
- lib/hnco/stop-watch.hh

5.120 SummationCancellation Class Reference

Summation cancellation.

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

Inheritance diagram for SummationCancellation:



Public Member Functions

- SummationCancellation (int n)

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- bool has_known_maximum ()

  Check for a known maximum.
- double get_maximum ()

  Get the global maximum.

Protected Member Functions

- void convert (const bit_vector_t &x)

  Convert a bit vector into a real vector.

Protected Attributes

- int _bv_size

  Bit vector size.
- std::vector< double > _buffer

  Buffer.

5.120.1 Detailed Description

Summation cancellation.

Encoding of a signed integer:

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

Reference:

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

Definition at line 48 of file cancellation.hh.

5.120.2 Constructor & Destructor Documentation

5.120.2.1 SummationCancellation()

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

Constructor.

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

Parameters

n	Size of the bit vector

Definition at line 71 of file cancellation.hh.

5.120.3 Member Function Documentation

5.120.3.1 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 87 of file cancellation.hh.

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

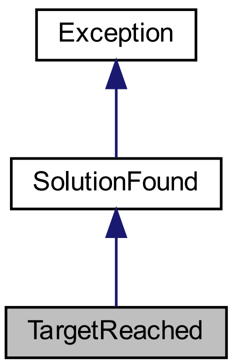
- lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

5.121 TargetReached Class Reference

Target reached.

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

Inheritance diagram for TargetReached:



Public Member Functions

- TargetReached (const algorithm::solution_t &solution)
    Constructor.

Additional Inherited Members

5.121.1 Detailed Description

Target reached.

Definition at line 71 of file exception.hh.

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

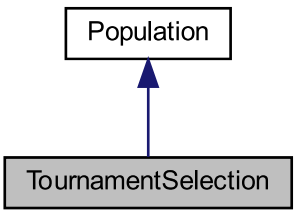
- lib/hnco/exception.hh

5.122 TournamentSelection Class Reference

Population with tournament selection

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

Inheritance diagram for TournamentSelection:



Public Member Functions

- TournamentSelection (int population_size, int n)
    Constructor.
- const bit_vector_t & select ()
    Selection.

Setters

- void set_tournament_size (int x)
    Set the tournament size.

Private Attributes

- std::uniform_int_distribution< int > _choose_individual

  Random index.

Parameters

- int _tournament_size = 10

  Tournament size.

Additional Inherited Members

5.122.1 Detailed Description

Population with tournament selection

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

5.122.2 Member Function Documentation

5.122.2.1 select()

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

Selection.

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

Precondition

The population must be evaluated.

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

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

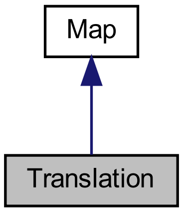
- lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

5.123 Translation Class Reference

Translation.

`#include <hnco/map.hh>`

Inheritance diagram for Translation:



Public Member Functions

- void random (int n)

  Random instance.
- void map (const bit_vector_t &input, bit_vector_t &output)

  Map
- int get_input_size ()

  Get input size.
- int get_output_size ()

  Get output size.
- bool is_surjective ()

  Check for surjective map.
- void set_bv (const bit_vector_t &bv)

  Set the translation vector.

Private Member Functions

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

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

  Load.

Private Attributes

- bit_vector_t _bv

  Translation vector

Friends

- class **boost::serialization::access**

5.123.1 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 74 of file map.hh.

5.123.2 Member Function Documentation

5.123.2.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 125 of file map.hh.

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

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

5.124 Transvection Struct Reference

Transvection.

```
#include <hnco/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.

5.124.1 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 62 of file transvection.hh.

5.124.2 Member Function Documentation

5.124.2.1 is_valid()

```
bool is_valid (
            int n ) const
```

Check validity.

Parameters

n	Dimension

Definition at line 46 of file transvection.cc.

5.124.2.2 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 115 of file transvection.cc.

5.124.2.3 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 103 of file transvection.cc.

5.124.2.4 random()

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

Sample a random transvection.

Parameters

n	Dimension

Precondition

$n > 1$

Definition at line 59 of file transvection.cc.

5.124.2.5 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 75 of file transvection.cc.

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

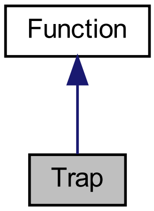
- lib/hnco/transvection.hh
- lib/hnco/transvection.cc

5.125 Trap Class Reference

Trap.

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

Inheritance diagram for Trap:



Public Member Functions

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

Private Attributes

- int _bv_size
  Bit vector size.
- int _num_traps
  Number of traps.
- int _trap_size
  Trap size.

5.125.1 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.

5.125.2 Constructor & Destructor Documentation

5.125.2.1 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.

5.125.3 Member Function Documentation

5.125.3.1 get_maximum()

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

Get the global maximum.

Returns

_bv_size

Reimplemented from Function.

Definition at line 88 of file trap.hh.

5.125.3.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 84 of file trap.hh.

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

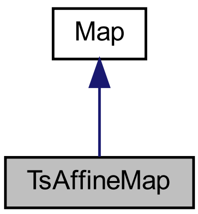
- lib/hnco/functions/trap.hh
- lib/hnco/functions/trap.cc

5.126 TsAffineMap Class Reference

Transvection sequence affine map.

`#include <hnco/map.hh>`

Inheritance diagram for TsAffineMap:



Public Types

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

  Sampling mode.

Public Member Functions

- void random (int n, int t, SamplingMode mode)
  Random instance.
- void map (const bit_vector_t &input, bit_vector_t &output)
  Map
- int get_input_size ()
  Get input size.
- int get_output_size ()
  Get output size.
- bool is_surjective ()
  Check for surjective map.
- void display (std::ostream &stream)
  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.

Private Attributes

- transvection_sequence_t _ts
  Transvection sequence
- bit_vector_t _bv
  Translation vector

Friends

- class **boost::serialization::access**

5.126.1 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 511 of file map.hh.

5.126.2 Member Enumeration Documentation

5.126.2.1 SamplingMode

```
enum SamplingMode
```

Sampling mode.

Enumerator

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

Definition at line 548 of file map.hh.

5.126.3 Member Function Documentation

5.126.3.1 is_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 591 of file map.hh.

5.126.3.2 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 185 of file map.cc.

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

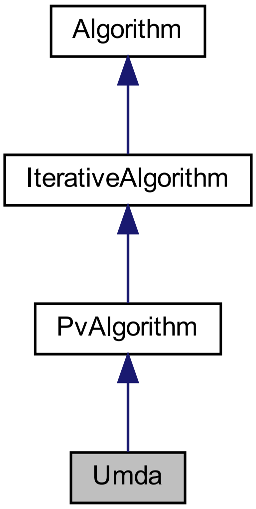
- lib/hnco/map.hh
- lib/hnco/map.cc

5.127 Umda Class Reference

Univariate marginal distribution algorithm.

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

Inheritance diagram for Umda:



Public Member Functions

- Umda (int n, int population_size)

  Constructor.

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.

5.127.1 Detailed Description

Univariate marginal distribution algorithm.

Reference:

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

Definition at line 41 of file umda.hh.

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

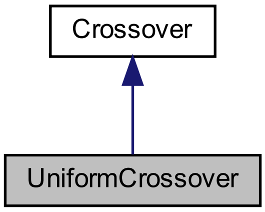
- lib/hnco/algorithms/pv/umda.hh
- lib/hnco/algorithms/pv/umda.cc

5.128 UniformCrossover Class Reference

Uniform crossover.

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

Inheritance diagram for UniformCrossover:



Public Member Functions

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

5.128.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

5.128.2 Member Function Documentation

5.128.2.1 breed()

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

Breed.

The offspring is the uniform crossover of two parents.

Parameters

parent1	First parent
parent2	Second parent
offspring	Offspring

Implements Crossover.

Definition at line 30 of file crossover.cc.

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

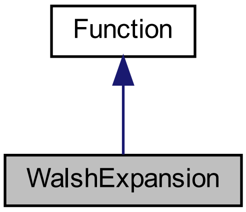
- lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

5.129 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



Public Member Functions

- WalshExpansion ()

  Constructor.
- int get_bv_size ()

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

  Evaluate a bit vector.
- void display (std::ostream &stream)

  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.

Private Member Functions

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

  Save.

Private Attributes

- std::vector< function::WalshTerm > _terms

  Terms.

Friends

- class **boost::serialization::access**

5.129.1 Detailed Description

Walsh expansion.

Its expression is of the form

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

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

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

5.129.2 Member Function Documentation

5.129.2.1 generate()

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

Instance generator.

Parameters

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

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

5.129.2.2 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 113 of file walsh-expansion.hh.

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

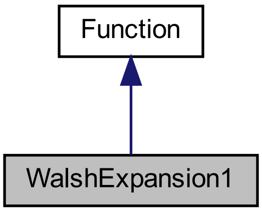
- lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

5.130 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

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

Inheritance diagram for WalshExpansion1:



Public Member Functions

- WalshExpansion1 ()
  Constructor.

Instance generators

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

Evaluation

- double evaluate (const bit_vector_t &)
  Evaluate a bit vector.
- double evaluate_incrementally (const bit_vector_t &x, double v, const hnco::sparse_bit_vector_t &flipped_bits)
  Incrementally evaluate a bit vector.

Information about the function

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

Private Member Functions

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

    Serialize.

Private Attributes

- std::vector< double > _linear

    Linear part.

Friends

- class **boost::serialization::access**

5.130.1 Detailed Description

Walsh expansion of degree 1.

Its expression is of the form

$f(x) = \sum_i a_i (1 - 2x_i)$

or equivalently

$f(x) = \sum_i a_i (-1)^{x_i}$

Definition at line 50 of file walsh-expansion-1.hh.

5.130.2 Member Function Documentation

5.130.2.1 generate()

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

Instance generator.

Parameters

n	Size of bit vectors
generator	Weight generator

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

5.130.2.2 has_known_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

5.130.2.3 provides_incremental_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

5.130.2.4 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 97 of file walsh-expansion-1.hh.

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

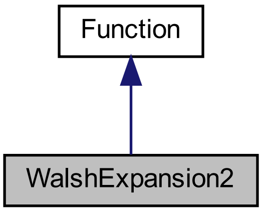
- lib/hnco/functions/walsh/walsh-expansion-1.hh
- lib/hnco/functions/walsh/walsh-expansion-1.cc

5.131 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

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

Inheritance diagram for WalshExpansion2:



Public Member Functions

- WalshExpansion2 ()

    Constructor.
- int get_bv_size ()

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

    Evaluate a bit vector.

Instance generators

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

    Instance generators.
- void random (int n)

    Instance generator.
- void generate_ising1_long_range (int n, double alpha)

    Generate one dimensional Ising model with long range interactions.
- void generate_ising1_long_range_periodic (int n, double alpha)

    Generate one dimensional Ising model with long range interactions and periodic boundary conditions.

Private Member Functions

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

    Serialize.
- void resize (int n)

    Resize data structures.

Private Attributes

- std::vector< double > _linear

  Linear part.
- std::vector< std::vector< double > > _quadratic

  Quadratic part.

Friends

- class **boost::serialization::access**

5.131.1 Detailed Description

Walsh expansion of degree 2.

Its expression is of the form

$f(x) = \sum_i a_i(1-2x_i) + \sum_{i<j} a_{ij}(1-2x_i)(1-2x_j)$

or equivalently

$f(x) = \sum_i a_i(-1)^{x_i} + \sum_{i<j} a_{ij}(-1)^{x_i+x_j}$

Definition at line 50 of file walsh-expansion-2.hh.

5.131.2 Member Function Documentation

5.131.2.1 generate()

```
void generate (
            int n,
            LinearGen linear_gen,
            QuadraticGen quadratic_gen )  [inline]
```

Instance generators.

Parameters

n	Size of bit vectors
linear_gen	Generator for the linear part
quadratic_gen	Generator for the quadratic part

Definition at line 95 of file walsh-expansion-2.hh.

5.131.2.2 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.

5.131.2.3 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.

5.131.2.4 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 117 of file walsh-expansion-2.hh.

5.131.3 Member Data Documentation

5.131.3.1 _quadratic

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

Quadratic part.

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

Definition at line 73 of file walsh-expansion-2.hh.

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

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

5.132 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.

5.132.1 Detailed Description

Walsh transform term.

Definition at line 35 of file walsh-term.hh.

5.132.2 Member Data Documentation

5.132.2.1 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