GraphSpace with C++

Generated by Doxygen 1.8.17

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 distance< T > Class Template Reference

```
#include <distance.h>
```

Inheritance diagram for distance< T >:

3.2 Distance::distanceHolder< T > Class Template Reference

Public Member Functions

- **distanceHolder** (distances d_id)
- void **setDistance** (distances d_id)

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

- DistanceFactory.h

3.3 euclidean< T > Class Template Reference

```
#include <euclidean.h>
```

Inheritance diagram for euclidean< T >:

Collaboration diagram for euclidean< T >:

Public Member Functions

- T the_sim (attr_type< T > x, attr_type< T > y)
- T the_dis (attr_type< T > x, attr_type< T > y)
- T node_dis (attr_type< T > x, attr_type< T > y) override
- T **node_sim** (attr_type< T > x, attr_type< T > y) override
- T edge_dis (attr_type< T > x, attr_type< T > y) override
- T **edge_sim** (attr_type< T > x, attr_type< T > y) override
- std::string get_Instance () override

3.3.1 Detailed Description

template<class T>
class euclidean< T >

Class that inherit form the class distance and that implements the euclidean distance

3.3.2 Member Function Documentation

3.3.2.1 edge_dis()

```
template<class T >
T euclidean< T >::edge_dis (
             attr_type< T > x,
             attr_type< T > y )  [override], [virtual]
```

Method to compute the distance between two edges

Implements distance< T >.

3.3.2.2 get_Instance()

```
template<class T >
std::string euclidean< T >::get_Instance  [override], [virtual]
```

This method returns the name of the distance used

Implements distance< T >.

3.3.2.3 node_dis()

```
template<class T >
T euclidean< T >::node_dis (
             attr_type< T > x,
             attr_type< T > y )  [override], [virtual]
```

Method to compute the distance between two nodes

Implements distance< T >.

3.3.2.4 the_dis()

```
template<class T >
T euclidean< T >::the_dis (
             attr_type< T > x,
             attr_type< T > y )
```

function to compute the distance between two nodes or two edges.

3.3.2.5 the_sim()

```
template<class T >
T euclidean< T >::the_sim (
             attr_type< T > x,
             attr_type< T > y )
```

function to compute pointwise product

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

- euclidean.h

3.4 GA< T > Class Template Reference

```
#include <GA.h>
```

Inheritance diagram for GA< T >:

Collaboration diagram for GA< T >:

Public Member Functions

- GA ()=default
- GA (Distance::distances _d)
- void match (GraphPointer< T > first_graph, GraphPointer< T > second_graph) override
- bool isStable (GraphPointer< T > first_graph, GraphPointer< T > second_graph, MatrixXd M1, MatrixXd M2, double eps)
- void initializeMatchMatrix (GraphPointer< T > x, GraphPointer< T > y)
- void cleanup (GraphPointer< T > first_graph, GraphPointer< T > second_graph)
- void setAssociationGraph (GraphPointer< T > first_graph, GraphPointer< T > second_graph)

Additional Inherited Members

3.4.1 Detailed Description

template<class T>
class GA< T >

Class that implemets the Graduate assigned algorithm to match two graph

3.4.2 Constructor & Destructor Documentation

3.4.2.1 GA() [1/2]

```
template<class T >
GA< T >::GA ( )  [default]
```

Default constructor

3.4.2.2 GA() [2/2]

```
template<class T >
GA< T >::GA (
             Distance::distances _d )  [inline]
```

Constructor if you want to set a specific distance

3.4.3 Member Function Documentation

3.4.3.1 cleanup()

```
template<class T >
void GA< T >::cleanup (
             GraphPointer< T > first_graph,
             GraphPointer< T > second_graph )
```

accessory function used by the match method to compute the matching with the hungarian algorithm in the Munkres function

3.4.3.2 initializeMatchMatrix()

```
template<class T >
void GA< T >::initializeMatchMatrix (
             GraphPointer< T > x,
             GraphPointer< T > y )
```

accessory function used by the match method to setup all the variables needed

3.4.3.3 isStable()

```
template<class T >
bool GA< T >::isStable (
            GraphPointer< T > first_graph,
            GraphPointer< T > second_graph,
            MatrixXd M1,
            MatrixXd M2,
            double eps )
```

accessory function used by the match method to controll that all is going well

3.4.3.4 match()

```
template<class T >
void GA< T >::match (
            GraphPointer< T > first_graph,
            GraphPointer< T > second_graph )  [override], [virtual]
```

Method that takes as input two GraphPointer and compute the permutation vector according with the Graduate assigned algorithm. Vector that is stored in the attribute f of the class Matcher

Implements matcher< T >.

3.4.3.5 setAssociationGraph()

```
template<class T >
void GA< T >::setAssociationGraph (
            GraphPointer< T > first_graph,
            GraphPointer< T > second_graph )
```

accessory function used by the match method to setup all the variables needed

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

- GA.h

3.5 gpc< T > Class Template Reference

Public Member Functions

- gpc (const GraphSet< T > &_gs)
- GraphSet< T > get_gs () const
- Eigen::Matrix< T, Eigen::Dynamic, 1 > get_barycenter () const
- GraphPointer< T > get_barycenter_net () const
- void set_barycenter (Eigen::Matrix< T, Eigen::Dynamic, Eigen::Dynamic >)
- std::tuple< Eigen::MatrixXd, Eigen::MatrixXd, Eigen::VectorXd > est_pc (int n_comp, bool scale)
- void align_geo (const geodesic< T > &geo, bool scale, int s_min, int s_max)
- std::tuple< Eigen::MatrixXd, std::vector< std::map< std::pair< int, int >, attr_type< double > > >, Eigen::VectorXd > gpc_aac (int max_iterations, double tol, int n_comp, bool scale, double s_min, double s_max)
- void **set_barycenter** (Eigen::Matrix< int, Eigen::Dynamic, Eigen::Dynamic > Mat)
- void **set_barycenter** (Eigen::Matrix< float, Eigen::Dynamic, Eigen::Dynamic > Mat)
- void **set_barycenter** (Eigen::Matrix< double, Eigen::Dynamic, Eigen::Dynamic > Mat)

3.5.1 Constructor & Destructor Documentation

3.5.1.1 gpc()

```
template<class T >
gpc< T >::gpc (
            const GraphSet< T > & _gs )  [inline]
```

Constructor

3.5.2 Member Function Documentation

3.5.2.1 align_geo()

```
template<class T >
void gpc< T >::align_geo (
            const geodesic< T > & geo,
            bool scale,
            int s_min,
            int s_max )
```

This method aligns the graphset with respect to a given geodesic

3.5.2.2 est_pc()

```
template<class T >
std::tuple< Eigen::MatrixXd, Eigen::MatrixXd, Eigen::VectorXd > gpc< T >::est_pc (
            int n_comp,
            bool scale )
```

This method estimate the PCA with respect to a given alignment of the graphset. It is only an estimation, it is not the optimal one.

3.5.2.3 get_barycenter()

```
template<class T >
Eigen::Matrix< T, Eigen::Dynamic, 1 > gpc< T >::get_barycenter
```

Getter for the barycenter

3.5.2.4 get_barycenter_net()

```
template<class T >
GraphPointer< T > gpc< T >::get_barycenter_net
```

Getter for the barycenter_net

3.5.2.5 get_gs()

```
template<class T >
GraphSet< T > gpc< T >::get_gs
```

Getter for the graphset

3.5.2.6 gpc_aac()

```
template<class T >
std::tuple< Eigen::MatrixXd, std::vector< std::map< std::pair< int, int >, attr_type< double
> > >, Eigen::VectorXd > gpc< T >::gpc_aac (
             int max_iterations,
             double tol,
             int n_comp,
             bool scale,
             double s_min,
             double s_max )
```

This method compute the Geodesic Principal Components of the graphset with the Align All and Compute principle. It takes in input the maximum number of iterations(max_iterations), the tollerance (tol), the number of principal components wanted to be estimated (n_comp), a flag to indicate if you want to scale the PCA (scale) and the begin and end positionof the geodesic (?)

3.5.2.7 set_barycenter()

```
template<class T >
void gpc< T >::set_barycenter (
             Eigen::Matrix< T, Eigen::Dynamic, Eigen::Dynamic >  )
```

Setter of the barycenter

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

- gpc.h

3.6 Graph< T > Class Template Reference

```
#include <Graph.h>
```

Public Member Functions

- Graph (bool _oriented)
- Graph (const std::map< std::pair< int, int >, attr_type< T >> &_graph_map, const bool _oriented)
- Graph (std::map< std::pair< std::pair< int, int >, std::pair< int, int >>, double > product_graph_↩ constructor, bool oriented)
- void add_vertex (const attr_type< T > &attribute, const int id_vertex)
- void add_edge (const int id_vertex1, const int id_vertex2, const attr_type< T > &edge_attribute)
- bool is_oriented () const
- bool isempty () const
- int get_n_nodes () const
- std::list< int > get_vertices_id () const
- std::map< std::pair< int, int >, attr_type< T > > get_graph_map () const
- int get_vertex_size () const
- int get_edge_size () const
- std::vector< std::vector< int > > get_adj () const
- void construct_adj ()
- GraphPointer< T > permute (const std::vector< int > &f) const
- void print_map () const
- std::set< std::pair< int, int > > get_keys () const
- void grow (int size, const attr_type< T > &new_attribute)
- GraphPointer< T > scale (double a) const

3.6.1 Detailed Description

template<class T>
class Graph< T >

Class Graph is used to define a graph object

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Graph() [1/3]

```
template<class T >
Graph< T >::Graph (
            bool _oriented )  [inline]
```

Constructor

3.6.2.2 Graph() [2/3]

```
template<class T >
Graph< T >::Graph (
            const std::map< std::pair< int, int >, attr_type< T >> & _graph_map,
            const bool _oriented )
```

Constructor

3.6.2.3 Graph() [3/3]

```
template<class T >
Graph< T >::Graph (
            std::map< std::pair< std::pair< int, int >, std::pair< int, int >>, double >
product_graph_constructor,
            bool oriented )
```

Constructor

3.6.3 Member Function Documentation

3.6.3.1 add_edge()

```
template<class T >
void Graph< T >::add_edge (
            const int id_vertex1,
            const int id_vertex2,
            const attr_type< T > & edge_attribute )
```

The method adds an edge to the graph

3.6.3.2 add_vertex()

```
template<class T >
void Graph< T >::add_vertex (
            const attr_type< T > & attribute,
            const int id_vertex )
```

The methods adds a vertex to the graph

3.6.3.3 construct_adj()

```
template<class T >
void Graph< T >::construct_adj
```

The method constructs the adjacency list of the graph and it stores the matrix in the attribute

3.6.3.4 get_adj()

```
template<class T >
std::vector< std::vector< int > > Graph< T >::get_adj
```

The method returns the adjacency list of the graph

3.6.3.5 get_edge_size()

```
template<class T >
int Graph< T >::get_edge_size
```

The method returns the size of the edge attribute

3.6.3.6 get_graph_map()

```
template<class T >
std::map< std::pair< int, int >, attr_type< T > > Graph< T >::get_graph_map
```

The methods returns the map that describes the graph

3.6.3.7 get_keys()

```
template<class T >
std::set< std::pair< int, int > > Graph< T >::get_keys
```

The methods returns a set that contains all the keys of the graph map, namely all the nodes couple present in the graph

3.6.3.8 get_n_nodes()

```
template<class T >
int Graph< T >::get_n_nodes
```

The methods return the number of the vertices of the graph

3.6.3.9 get_vertex_size()

```
template<class T >
int Graph< T >::get_vertex_size
```

The method returns the size of the vertex attribute

3.6.3.10 get_vertices_id()

```
template<class T >
std::list< int > Graph< T >::get_vertices_id
```

The methods returns a list of the graph vertices id

3.6.3.11 grow()

```
template<class T >
void Graph< T >::grow (
             int size,
             const attr_type< T > & new_attribute )
```

The method increases the size of the graph creating new vertex with a specified input, until the chosen size is reached. The new vertex created are not linked with other vertex already existing

3.6.3.12 is_oriented()

```
template<class T >
bool Graph< T >::is_oriented
```

The methods returns the orientation of the graph

3.6.3.13 isempty()

```
template<class T >
bool Graph< T >::isempty
```

The methods returns a bool that indicates if the graph is empty

3.6.3.14 permute()

```
template<class T >
GraphPointer< T > Graph< T >::permute (
             const std::vector< int > & f ) const
```

The method returns the permuted graph given the permutation to apply

3.6.3.15 print_map()

```
template<class T >
void Graph< T >::print_map
```

The method prints the graph in the map form

3.6.3.16 scale()

```
template<class T >
GraphPointer< T > Graph< T >::scale (
             double a ) const
```

The method return a graph that has got multiplied attribute by the input constant

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

- gpc.h
- Graph.h

3.7 GraphSet< T > Class Template Reference

```
#include <GraphSet.h>
```

Public Member Functions

- **GraphSet** (bool _oriented)
- GraphSet (const std::vector< std::map< std::pair< int, int >, attr_type< T >>> &graph_maps, const bool orientation)
- GraphSet (const std::vector< std::map< std::pair< int, int >, attr_type< T >>> &graph_maps, const bool orientation, const Matcher::matchers _m, const Distance::distances _d)
- bool is_oriented () const
- void add_graph (GraphPointer< T > graph)
- std::vector< GraphPointer< T > > get_graphset () const
- std::vector< std::map< std::pair< int, int >, attr_type< T > > > get_graphset_maps () const
- std::vector< GraphPointer< T > > get_aligned_GraphSet () const
- std::vector< std::map< std::pair< int, int >, attr_type< T > > > get_aligned_GraphSet_maps () const
- std::vector< std::vector< int > > get_permutation_vector () const
- void set_permutation_vector (int index, std::vector< int > p)
- GraphPointer< T > get_mean () const
- Matcher::matchers get_matcher () const
- void set_match (const Matcher::matchers _m)
- Distance::distances get_distance () const
- void set_distance (const Distance::distances _d)
- int get_n_max () const
- int get_v_attr_max () const
- int get_e_attr_max () const
- void align (GraphPointer< T > g)
- GraphSet< T > permuted_graphset () const
- void save_aligned ()
- Eigen::Matrix< T, Eigen::Dynamic, Eigen::Dynamic > to_matrix_with_attr (bool aligned) const
- GraphPointer< T > est (const GraphPointer< T > &m1) const
- void mean_aac (int max_iteration, double tol)
- void read_from_text (std::string file_name)

3.7.1 Detailed Description

template<class T>
class GraphSet< T >

Class GraphSet defines a set of graph. template T defines the type of data that is contained in the attributes A set of graph is defined as a vector of pointers to graph objects

3.7.2 Constructor & Destructor Documentation

3.7.2.1 GraphSet() [1/2]

```
template<class T >
GraphSet< T >::GraphSet (
             const std::vector< std::map< std::pair< int, int >, attr_type< T >>> & graph_↩
maps,
             const bool orientation )
```

Constructor

3.7.2.2 GraphSet() [2/2]

```
template<class T >
GraphSet< T >::GraphSet (
             const std::vector< std::map< std::pair< int, int >, attr_type< T >>> & graph_↩
maps,
             const bool orientation,
             const Matcher::matchers _m,
             const Distance::distances _d )
```

Constructor

3.7.3 Member Function Documentation

3.7.3.1 add_graph()

```
template<class T >
void GraphSet< T >::add_graph (
             GraphPointer< T > graph )
```

The method adds a graph to the GraphSet. A graph can be added to the graphset only if it has the same orientation as the graphset.

3.7.3.2 align()

```
template<class T >
void GraphSet< T >::align (
             GraphPointer< T > g )
```

The methods saves in the permutation_vector the permutations that have to be done to align the GraphSet with the specified input

3.7.3.3 est()

```
template<class T >
GraphPointer< T > GraphSet< T >::est (
             const GraphPointer< T > & m1 ) const
```

The methods return an estimate, but not the final value, of the mean of the GraphSet

3.7.3.4 get_aligned_GraphSet()

```
template<class T >
std::vector< GraphPointer< T > > GraphSet< T >::get_aligned_GraphSet
```

The method returns the vector of aligned graphs

3.7.3.5 get_aligned_GraphSet_maps()

```
template<class T >
std::vector< std::map< std::pair< int, int >, attr_type< T > > > GraphSet< T >::get_↩
aligned_GraphSet_maps
```

The method returns a vector of maps, related to the aligned graphs

3.7.3.6 get_distance()

```
template<class T >
Distance::distances GraphSet< T >::get_distance
```

The method returns the enum correspondent to the distance used in the Graphset

3.7.3.7 get_e_attr_max()

```
template<class T >
int GraphSet< T >::get_e_attr_max
```

The methods returns the maximum dimension of edge attributes among all graphs

3.7.3.8 get_graphset()

```
template<class T >
std::vector< GraphPointer< T > > GraphSet< T >::get_graphset
```

The method returns the vector of graph pointers

3.7.3.9 get_graphset_maps()

```
template<class T >
std::vector< std::map< std::pair< int, int >, attr_type< T > > > GraphSet< T >::get_↩
graphset_maps
```

The method returns a vector of maps, related to the graphs contained in the GraphSet

3.7.3.10 get_matcher()

```
template<class T >
Matcher::matchers GraphSet< T >::get_matcher
```

The method returns the enum correspondent to the matcher used in the Graphset

3.7.3.11 get_mean()

```
template<class T >
GraphPointer< T > GraphSet< T >::get_mean
```

The method returns the mean of the GraphSet

3.7.3.12 get_n_max()

```
template<class T >
int GraphSet< T >::get_n_max
```

The method returns the maximum number of nodes that a graph in the GraphSet has.

3.7.3.13 get_permutation_vector()

```
template<class T >
std::vector< std::vector< int > > GraphSet< T >::get_permutation_vector
```

The method returns the vector of permutation

3.7.3.14 get_v_attr_max()

```
template<class T >
int GraphSet< T >::get_v_attr_max
```

The methods returns the maximum dimension of vertex attributes among all graphs

3.7.3.15 is_oriented()

```
template<class T >
bool GraphSet< T >::is_oriented
```

The methods return a bool that indicates if the graphs in the graphset are orinted or not

3.7.3.16 mean_aac()

```
template<class T >
void GraphSet< T >::mean_aac (
             int max_iteration,
             double tol )
```

The methods returns the Fréchet Mean of the GraphSet

3.7.3.17 permuted_graphset()

```
template<class T >
GraphSet< T > GraphSet< T >::permuted_graphset
```

The method returns the permuted graphset correspondent to the permutation vector

3.7.3.18 read_from_text()

```
template<class T >
void GraphSet< T >::read_from_text (
             std::string file_name )
```

This methods reads a graph from a corrected formatted text file

3.7.3.19 save_aligned()

```
template<class T >
void GraphSet< T >::save_aligned
```

The method permutes the graphset and saves the permuted graphs obtained in the aligned_graphset

3.7.3.20 set_distance()

```
template<class T >
void GraphSet< T >::set_distance (
             const Distance::distances _d )
```

The method sets the enum of the distance

3.7.3.21 set_match()

```
template<class T >
void GraphSet< T >::set_match (
             const Matcher::matchers _m )
```

The methods set the enum of the matcher

3.7.3.22 set_permutation_vector()

```
template<class T >
void GraphSet< T >::set_permutation_vector (
             int index,
             std::vector< int > p )
```

The method allows to modify one component of the permutation vector with the permutation given as input

3.7.3.23 to_matrix_with_attr()

```
template<class T >
Eigen::Matrix< T, Eigen::Dynamic, Eigen::Dynamic > GraphSet< T >::to_matrix_with_attr (
             bool aligned ) const
```

The methods return a matrix associated to the graphset Every row corresponds to a graph Every columns correspond to a connection of the graph

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

- GraphSet.h

3.8 ID< T > Class Template Reference

```
#include <ID.h>
```

Inheritance diagram for ID< T >:

Collaboration diagram for ID< T >:

Public Member Functions

- ID ()=default
- ID (Distance::distances _d)
- void match (GraphPointer< T > first_graph, GraphPointer< T > second_graph) override

Additional Inherited Members

3.8.1 Detailed Description

template<class T>
class ID< T >

Class that implements the ID matcher

3.8.2 Constructor & Destructor Documentation

3.8.2.1 ID() [1/2]

```
template<class T >
ID< T >::ID ( )  [default]
```

Default constructor

3.8.2.2 ID() [2/2]

```
template<class T >
ID< T >::ID (
             Distance::distances _d )  [inline]
```

Constructor if you want to set a specific distance

3.8.3 Member Function Documentation

3.8.3.1 match()

```
template<class T >
void ID< T >::match (
            GraphPointer< T > first_graph,
            GraphPointer< T > second_graph )  [override], [virtual]
```

Method that takes as input two GraphPointer and compute the permutation vector that is the identity. Vector that is stored in the attribute f of the class Matcher

Implements matcher< T >.

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

- ID.h

3.9 matcher< T > Class Template Reference

```
#include <matcher.h>
```

Inheritance diagram for matcher< T >:

Public Member Functions

- matcher ()
- matcher (Distance::distances _distance)
- virtual void match (GraphPointer< T > first_graph, GraphPointer< T > second_graph)=0
- double the_dis (GraphPointer< T > X, GraphPointer< T > Y)
- std::string get_distance ()
- std::vector< int > get_f ()
- double get_dist ()
- void set_dist (Distance::distances _d)

Protected Attributes

- double dist
- std::vector< int > f
- DistancePointer< T > distance

3.9.1 Detailed Description

template<class T>
class matcher< T >

Base abstruct class for matchers

3.9.2 Constructor & Destructor Documentation

3.9.2.1 matcher() [1/2]

```
template<class T >
matcher< T >::matcher ( )  [inline]
```

Constructor

3.9.2.2 matcher() [2/2]

```
template<class T >
matcher< T >::matcher (
             Distance::distances _distance )  [inline]
```

Constructor

3.9.3 Member Function Documentation

3.9.3.1 get_dist()

```
template<class T >
double matcher< T >::get_dist
```

Getter for dist

3.9.3.2 get_distance()

```
template<class T >
std::string matcher< T >::get_distance
```

The method returns the name of the distance used

3.9.3.3 get_f()

```
template<class T >
std::vector< int > matcher< T >::get_f
```

Getter for f

3.9.3.4 match()

```
template<class T >
virtual void matcher< T >::match (
            GraphPointer< T > first_graph,
            GraphPointer< T > second_graph ) [pure virtual]
```

The methods matches two graphs, saving in f the best permutation

Implemented in GA< T >, and ID< T >.

3.9.3.5 set_dist()

```
template<class T >
void matcher< T >::set_dist (
            Distance::distances _d )
```

Setter of distance pointer

3.9.3.6 the_dis()

```
template<class T >
double matcher< T >::the_dis (
            GraphPointer< T > X,
            GraphPointer< T > Y )
```

The methods sets dist equal to distance of the graphs X and Y based on the permutation f. In order to have the right distance, you have to be sure that in f there is the right permutation vector, so before running this function you have to do the matching.

3.9.4 Member Data Documentation

3.9.4.1 dist

```
template<class T >
double matcher< T >::dist [protected]
```

When match method is called, in this variable is stored the distance between the two matched graphs

3.9.4.2 distance

```
template<class T >
DistancePointer<T> matcher< T >::distance [protected]
```

Poiter to the distance that is used in the match

3.9.4.3 f

```
template<class T >
std::vector<int> matcher< T >::f  [protected]
```

Permutation of X to get close to Y

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

- matcher.h

3.10 Matcher::matcherHolder< T > Class Template Reference

Public Member Functions

- **matcherHolder** (matchers m_id)
- void **setMatcher** (matchers m_id)

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

- MatcherFactory.h

3.11 Munkres< T > Class Template Reference

Public Member Functions

- Matrix< T > **pad_matrix** (Matrix< T > &Matrix, T pad_value)
- std::vector< std::pair< int, int > > **compute** (Matrix< T > cost_matrix)
- Matrix< int > **_make_matrix** (int n, T value)
- void **_clear_covers** ()
- std::pair< int, int > **_find_a_zero** (int i0, int j0)
- T **_find_smallest** ()
- int **_find_star_in_row** (int row)
- int **_find_star_in_col** (int col)
- int **_find_prime_in_row** (int row)
- void **_convert_path** (std::vector< std::vector< int >> _path, int count)
- void **_erase_primes** ()
- int **step1** ()
- int **step2** ()
- int **step3** ()
- int **step4** ()
- int **step5** ()
- int **step6** ()

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

- Munkres.h

Index