BIRCH Clustering

The C++ implmentation for BIRCH clustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

BIRCH Clustering Algorithm

BIRCH is a linear time scanning implemented by B+ tree to merge all the candidates within the distance threshold into one cluster. There are several comments on this BIRCH implementation

- The original implementation has hardcoded dimension size, and it **needs to be revised** according to the input data sets
- The final clusters are **spherical shape** because of squared distance threshold is utilized
- It does not guarantee the clustering result will be meaningful since the final clustering result heavily relies on the input distance threshold
- In order for fair comparison with those clustering techniques with an input of cluster number, we developed a **binary search** algorithm with input of cluster numbers to decide the possible distance threshold
  - For spatial similarity measures, it can almost guarantee the **distance threshold** can be found to generate **approximate required number of clusters**
  - For some similarity measures, it can never find such distance threshold, hence we set the *max search iteration* to be 10

Number of clusters as input

The program supports two kinds of input for number of clusters

- Direct input after the query information $>$ Input cluster number among [0, 1000] for norm X:
- Read the cluster numbers from a txt file
  - The txt file is called 'cluster_number' in the /dataset/ folder
  - The 'cluster number' has the following format
    0:10 // for similarity measure 0, the input of cluster number is 10 1:10 // for similarity measure 1, the input of cluster number is 10 2:10 4:10 12:10 13:10 14:10 15:10 17:10
    for better batch processing especially in our experiment when the code is automatically run on the server

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

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 CFEntry< dim > Class Template Reference

`#include <CFEntry.h>`

Public Types

- enum { fdim = dim }
- typedef boost::shared_ptr< CFNode< dim > > cfnode_sptr_type
- typedef std::pair< CFEntry *, CFEntry * > cfentry_pair_type
- typedef std::vector< CFEntry * > cfentry_ptr_vec_type
- typedef float_type(* dist_func_type) (const CFEntry &, const CFEntry &)
- typedef std::vector< CFEntry > cfentry_vec_type

Public Member Functions

- CFEntry ()
- CFEntry (const CFEntry &other)
- template<typename T >
  CFEntry (T *item)
- template<typename T >
  CFEntry (T *item, std::size_t id)
- CFEntry (const cfnode_sptr_type &in_child)
- CFEntry operator+ (const CFEntry &rhs)
- void operator+= (const CFEntry &e)
- void operator-= (const CFEntry &e)
- bool operator== (const CFEntry &e) const
- bool HasChild () const

Public Attributes

- std::size_t n
- float_type sum [dim]
- float_type sum_sq
- cfnode_sptr_type child
- std::size_t id

5.1.1 Detailed Description

template<boost::uint32_t dim>
class CFEntry< dim >

CFEntry is compact representation of group of data-points. This entry contains linear sums each dimension and one square sum to represent data-points in this group

Definition at line 50 of file CFEntry.h.

5.1.2 Member Typedef Documentation

5.1.2.1 template<boost::uint32_t dim> typedef std::pair<CFEntry*, CFEntry*> CFEntry< dim >::cfentry_pair_type

Definition at line 54 of file CFEntry.h.

5.1.2.2 template<boost::uint32_t dim> typedef std::vector<CFEntry*> CFEntry< dim >::cfentry_ptr_vec_type

pair cfentry pointers.

Definition at line 55 of file CFEntry.h.

5.1.2.3 template<boost::uint32_t dim> typedef std::vector<CFEntry> CFEntry< dim >::cfentry_vec_type

distance function pointer.

Definition at line 57 of file CFEntry.h.

5.1.2.4 template<boost::uint32_t dim> typedef boost::shared_ptr<CFNode<dim> > CFEntry< dim >::cfnode_sptr_type

Definition at line 53 of file CFEntry.h.

5.1.2.5 template<boost::uint32_t dim> typedef float_type(* CFEntry< dim >::dist_func_type) (const CFEntry &, const CFEntry &)

vector of cfentry pointers.

Definition at line 56 of file CFEntry.h.

5.1.3 Member Enumeration Documentation

5.1.3.1 template<boost::uint32_t dim> anonymous enum

vector of cfentries.

Enumerator

fdim

Definition at line 58 of file CFEntry.h.

```
58 { fdim = dim };
```

5.1.4 Constructor & Destructor Documentation

5.1.4.1 template<boost::uint32_t dim> CFEntry< dim >::CFEntry ( ) [inline]

enum for recognizing dimension outside this class. Empty construct initialized with zeros

Definition at line 60 of file CFEntry.h.

```
60                : n(0), sum_sq(0.0), id(0)
61     {
62         std::fill(sum, sum + dim, 0);
63     }
```

5.1.4.2 template<boost::uint32_t dim> CFEntry< dim >::CFEntry (const CFEntry< dim > & *other*) [inline]

Definition at line 64 of file CFEntry.h.

```
65     : n(other.n)
66     , sum_sq(other.n)
67     , child (other.child)
68     , id(0)
69     {
70         //std::fill(sum, sum + dim, 0);
71         std::copy(other.sum, other.sum + dim, sum);
72     }
```

5.1.4.3 template<boost::uint32_t dim> template<typename T > CFEntry< dim >::CFEntry (T * *item*) [inline]

Constructor when array of T type items come. initialize CFEntry with one data-point

Definition at line 85 of file CFEntry.h.

```
85                        : n(1), sum_sq(0.0), id(0)
86     {
87         std::copy( item, item + dim, sum );
88         for( std::size_t i = 0 ; i < dim ; i++ )
89             sum_sq += item[i] * item[i];
90     }
```

5.1.4.4 template<boost::uint32_t dim> template<typename T > CFEntry< dim >::CFEntry (T * *item*, std::size_t *id*) [inline]

Definition at line 93 of file CFEntry.h.

```
93                                      : n(1), sum_sq(0.0), id(id)
94     {
95         std::copy( item, item + dim, sum );
96         for( std::size_t i = 0 ; i < dim ; i++ )
97             sum_sq += item[i] * item[i];
98     }
```

5.1.4.5 template<boost::uint32_t dim> CFEntry< dim >::CFEntry (const cfnode_sptr_type & *in_child*) [inline]

Constructor for root entry with children

Definition at line 101 of file CFEntry.h.

```
101                                                 : n(0), sum_sq(0.0),
      child(in_child), id(0)
102     {
103         std::fill(sum, sum + dim, 0);
104     }
```

5.1.5 Member Function Documentation

5.1.5.1 template<boost::uint32_t dim> bool CFEntry< dim >::HasChild ( ) const [inline]

Does this CFEntry have children?

Definition at line 151 of file CFEntry.h.

```
151 { return child.get() != NULL; }
```

5.1.5.2 template<boost::uint32_t dim> CFEntry CFEntry< dim >::operator+ (const CFEntry< dim > & *rhs*) [inline]

Operator returning a new CFEntry merging from two CFEntries

Definition at line 107 of file CFEntry.h.

```
108     {
109         CFEntry e;
110         e.n = n + rhs.n;
111         for( std::size_t i = 0 ; i < dim ; i++ )
112             e.sum[i] = sum[i] + rhs.sum[i];
113         e.sum_sq = sum_sq + rhs.sum_sq;
114 
115         return e;
116     }
```

5.1.5.3 template<boost::uint32_t dim> void CFEntry< dim >::operator+= (const CFEntry< dim > & *e*) [inline]

Operator merging two CFEntries to the left-hand-side CFEntry

Definition at line 119 of file CFEntry.h.

```
120     {
121         for( std::size_t i = 0 ; i < dim ; i++ )
122         {
123             float_type val = e.sum[i];
124             sum[i] += val;
125             sum_sq += val*val;
126         }
127         n += e.n;
128     }
```

5.1.5.4 template<boost::uint32_t dim> void CFEntry< dim >::operator-= (const CFEntry< dim > & *e*) [inline]

Operator removing data-points from one CFEntry

Definition at line 131 of file CFEntry.h.

```
132     {
133         for( std::size_t i = 0 ; i < dim ; i++ )
134         {
135             float_type val = e.sum[i];
136             sum[i] -= val;
137             sum_sq -= val*val;
138         }
139         n -= e.n;
140     }
```

5.1.5.5 template<boost::uint32_t dim> bool CFEntry< dim >::operator== (const CFEntry< dim > & *e*) const [inline]

Definition at line 142 of file CFEntry.h.

```
143     {
144         if (n != e.n || sum_sq != e.sum_sq || child != e.
      child) return false;
145         for (int i = 0; i < dim; i++) if (sum[i] != e.sum[i]) return false;
146 
147         return true;
148     }
```

5.1.6 Member Data Documentation

5.1.6.1 template<boost::uint32_t dim> cfnode_sptr_type CFEntry< dim >::child

Definition at line 156 of file CFEntry.h.

5.1.6.2 template<boost::uint32_t dim> std::size_t CFEntry< dim >::id

Definition at line 157 of file CFEntry.h.

5.1.6.3 template<boost::uint32_t dim> std::size_t CFEntry< dim >::n

Definition at line 153 of file CFEntry.h.

5.1.6.4 template<boost::uint32_t dim> float_type CFEntry< dim >::sum[dim]

Definition at line 154 of file CFEntry.h.

5.1.6.5 template<boost::uint32_t dim> float_type CFEntry< dim >::sum_sq

Definition at line 155 of file CFEntry.h.

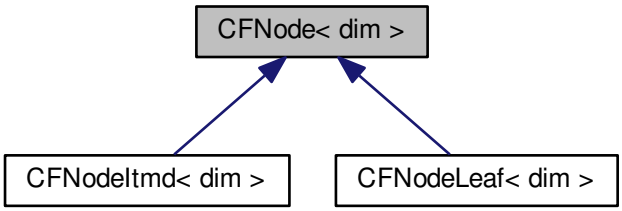
The documentation for this class was generated from the following file:

- CFEntry.h

5.2 CFNode< dim > Struct Template Reference

`#include <CFNode.h>`

Inheritance diagram for CFNode< dim >:



Public Member Functions

- CFNode ()
- virtual ~CFNode ()
- virtual bool IsLeaf () const =0
- void Add (CFEntry< dim > &e)
- void Replace (CFEntry< dim > &old_entry, CFEntry< dim > &new_entry)
- std::size_t MaxEntrySize () const
- bool IsFull () const
- bool IsEmpty () const

Public Attributes

- std::size_t size
- CFEntry< dim > entries [(PAGE_SIZE-(sizeof(CFNodeLeaf< dim > *)*2+sizeof(std::size_t)+sizeof(void *)))/sizeof(CFEntry< dim >)]

5.2.1 Detailed Description

template<boost::uint32_t dim>
struct CFNode< dim >

Definition at line 33 of file CFNode.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 template<boost::uint32_t dim> CFNode< dim >::CFNode ( ) [inline]

Definition at line 35 of file CFNode.h.

```
35              : size(0) {
36          //entries.resize((PAGE_SIZE - ( sizeof(CFNodeLeaf<dim> *)*2 /* 2 leaf node pointers */ +
        sizeof(std::size_t) /* size */ + sizeof(void*) /* vtptr */ )) / sizeof(CFEntry<dim> )/*max_entries*/);
37      }
```

5.2.2.2 template<boost::uint32_t dim> virtual CFNode< dim >::~CFNode ( ) [inline], [virtual]

Definition at line 43 of file CFNode.h.

```
43 {}
```

5.2.3 Member Function Documentation

5.2.3.1 template<boost::uint32_t dim> void CFNode< dim >::Add (CFEntry< dim > & *e*) [inline]

add new CFEntry to this CFNode

Definition at line 47 of file CFNode.h.

```
48      {
49  //        entries.push_back(e);
50  //        size++;
51          assert( size < MaxEntrySize() );
52          entries[size++] = e;
53      }
```

5.2.3.2 template<boost::uint32_t dim> bool CFNode< dim >::IsEmpty ( ) const [inline]

CFNode has nothing

Definition at line 81 of file CFNode.h.

```
82     {
83         return size == 0;
84     }
```

5.2.3.3 template<boost::uint32_t dim> bool CFNode< dim >::IsFull ( ) const [inline]

CFNode is full, no more CFEntries can be in

Definition at line 75 of file CFNode.h.

```
76     {
77         return size == MaxEntrySize();
78     }
```

5.2.3.4 template<boost::uint32_t dim> virtual bool CFNode< dim >::IsLeaf ( ) const [pure virtual]

Implemented in CFNodeItmd< dim >, and CFNodeLeaf< dim >.

5.2.3.5 template<boost::uint32_t dim> std::size_t CFNode< dim >::MaxEntrySize ( ) const [inline]

Max # of CFEntries this CFNode could contain

Definition at line 69 of file CFNode.h.

```
70     {
71         return ARRAY_COUNT(entries);
72     }
```

5.2.3.6 template<boost::uint32_t dim> void CFNode< dim >::Replace (CFEntry< dim > & *old_entry*, CFEntry< dim > & *new_entry*) [inline]

replace old CFEntry as new CFEntry

Definition at line 56 of file CFNode.h.

```
57     {
58         for (std::size_t i = 0; i < size; i++) {
59             if (entries[i] == old_entry) {
60                 entries[i] = new_entry;
61                 return;
62             }
63         }
64         // should not be reached here if replacement is successful
65         assert(false);
66     }
```

5.2.4 Member Data Documentation

5.2.4.1 template<boost::uint32_t dim> CFEntry<dim> CFNode< dim >::entries[(PAGE_SIZE-(sizeof(CFNodeLeaf< dim > *)*2+sizeof(std::size_t)+sizeof(void *)))/sizeof(CFEntry< dim >)]

CFEntries this CFNode contains

Definition at line 88 of file CFNode.h.

5.2.4.2 template<boost::uint32_t dim> std::size_t CFNode< dim >::size

Definition at line 86 of file CFNode.h.

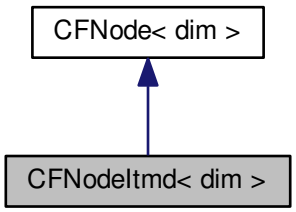
The documentation for this struct was generated from the following file:

- CFNode.h

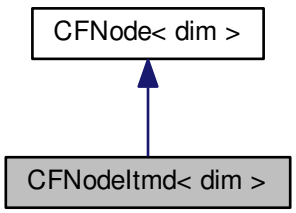
5.3 CFNodeItmd< dim > Struct Template Reference

`#include <CFNodeItmd.h>`

Inheritance diagram for CFNodeItmd< dim >:



Collaboration diagram for CFNodeItmd< dim >:



Public Member Functions

- CFNodeItmd ()
- virtual ~CFNodeItmd ()
- virtual bool IsLeaf () const

Additional Inherited Members

5.3.1 Detailed Description

template<boost::uint32_t dim>
struct CFNodeItmd< dim >

CFNode which is intermediate

Definition at line 14 of file CFNodeItmd.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 template<boost::uint32_t dim> CFNodeItmd< dim >::CFNodeItmd ( ) [inline]

Definition at line 16 of file CFNodeItmd.h.

```
16 : CFNode<dim>() {}
```

5.3.2.2 template<boost::uint32_t dim> virtual CFNodeItmd< dim >::~CFNodeItmd ( ) [inline], [virtual]

Definition at line 17 of file CFNodeItmd.h.

```
17 {};
```

5.3.3 Member Function Documentation

5.3.3.1 template<boost::uint32_t dim> virtual bool CFNodeItmd< dim >::IsLeaf ( ) const [inline], [virtual]

Implements CFNode< dim >.

Definition at line 18 of file CFNodeItmd.h.

```
18 { return false; }
```

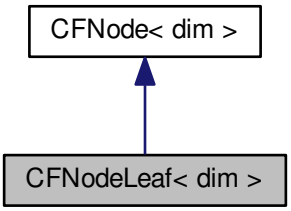
The documentation for this struct was generated from the following file:

- CFNodeItmd.h

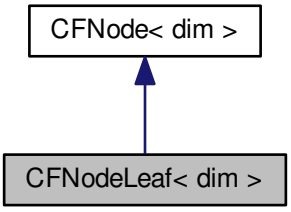
5.4 CFNodeLeaf< dim > Struct Template Reference

`#include <CFNode.h>`

Inheritance diagram for CFNodeLeaf< dim >:



Collaboration diagram for CFNodeLeaf< dim >:



Public Types

- typedef boost::shared_ptr< CFNode< dim > > cfnode_sptr_type

Public Member Functions

- CFNodeLeaf ()
- virtual ~CFNodeLeaf ()
- virtual bool IsLeaf () const

Public Attributes

- cfnode_sptr_type prev
- cfnode_sptr_type next

5.4.1 Detailed Description

template<boost::uint32_t dim>
struct CFNodeLeaf< dim >

CFNode which is leaf

Definition at line 31 of file CFNode.h.

5.4.2 Member Typedef Documentation

5.4.2.1 template<boost::uint32_t dim> typedef boost::shared_ptr<CFNode<dim> > CFNodeLeaf< dim >::cfnode_sptr_type

Definition at line 15 of file CFNodeLeaf.h.

5.4.3 Constructor & Destructor Documentation

5.4.3.1 template<boost::uint32_t dim> CFNodeLeaf< dim >::CFNodeLeaf ( ) [inline]

Definition at line 16 of file CFNodeLeaf.h.

```
16 : CFNode<dim>() {}
```

5.4.3.2 template<boost::uint32_t dim> virtual CFNodeLeaf< dim >::~CFNodeLeaf ( ) [inline], [virtual]

Definition at line 17 of file CFNodeLeaf.h.

```
17 {}
```

5.4.4 Member Function Documentation

5.4.4.1 template<boost::uint32_t dim> virtual bool CFNodeLeaf< dim >::IsLeaf ( ) const [inline], [virtual]

Implements CFNode< dim >.

Definition at line 18 of file CFNodeLeaf.h.

```
18 { return true; }
```

5.4.5 Member Data Documentation

5.4.5.1 template<boost::uint32_t dim> cfnode_sptr_type CFNodeLeaf< dim >::next

previous CFNode

Definition at line 21 of file CFNodeLeaf.h.

5.4.5.2 template<boost::uint32_t dim> cfnode_sptr_type CFNodeLeaf< dim >::prev

Definition at line 20 of file CFNodeLeaf.h.

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

- CFNode.h
- CFNodeLeaf.h

5.5 CFTree< dim > Class Template Reference

`#include <CFTree.h>`

Classes

- struct CFTreeInvalidItemSize
- struct CloseEntryLessThan
- struct HierarchicalClustering

Public Types

- enum { fdim = dim }
- typedef float float_type
- typedef std::vector< float_type > item_vec_type
- typedef boost::shared_ptr< CFNode< dim > > cfnode_sptr_type
- typedef std::pair< CFEntry< dim > *, CFEntry< dim > * > cfentry_pair_type
- typedef std::vector< CFEntry< dim > * > cfentry_ptr_vec_type
- typedef float_type(* dist_func_type) (const CFEntry< dim > &, const CFEntry< dim > &)
- typedef std::vector< CFEntry< dim > > cfentry_vec_type
- typedef boost::numeric::ublas::vector< float_type > ublas_vec_type
- typedef boost::numeric::ublas::symmetric_matrix< float_type > ublas_sym_matrix_type
- typedef std::vector< subcluster_summary > subsum_vec_type

Public Member Functions

- CFTree (float_type in_dist_threshold, std::size_t in_mem_limit, dist_func_type in_dist_func=_DistLarry, dist_func_type in_absorb_dist_func=_DistLarry)
- void setDistThreshold (const float_type &in_dist_threshold)
- ~CFTree (void)
- bool empty () const
- void insert (item_vec_type &item)
- template<typename T >
  void insert (T *item)
- void insert (CFEntry< dim > &e)
- leaf_iterator< dim > leaf_begin ()
- leaf_iterator< dim > leaf_end ()
- void get_entries (cfentry_vec_type &out_entries)
- void rebuild (bool extend=true)
- void redist (std::vector< item_type< 4824u > >::iterator begin, std::vector< item_type< 4824u > >::iterator end, std::vector< CFEntry< 4824u > > &entries, std::vector< int > &out_cid)
- void cluster (cfentry_vec_type &entries)

Static Public Member Functions

- static float_type _DistD0 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD1 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD2 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD3 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD4 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD8 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistD16 (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _DistLarry (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)
- static float_type _Diameter (const CFEntry< dim > &e)
- static float_type _Radius (const CFEntry< dim > &e)

Static Public Attributes

- static MetricPreparation object = MetricPreparation()
- static int normOption = -1
- static int totalNodes = 0

Private Member Functions

- void insert (CFNode< dim > *node, CFEntry< dim > &new_entry, bool &bsplit)
- CFEntry< dim > * find_close (CFNode< dim > *node, CFEntry< dim > &new_entry)
- void split (CFNode< dim > &node, CFEntry< dim > &close_entry, CFEntry< dim > &new_entry, bool &bsplit)
- void split_root (CFEntry< dim > &e)
- void rearrange (cfentry_ptr_vec_type &entries, cfentry_pair_type &far_pair, CFEntry< dim > &entry_lhs, CFEntry< dim > &entry_rhs)
- void find_farthest_pair (std::vector< CFEntry< dim > * > &entries, cfentry_pair_type &far_pair)
- float_type average_dist_closest_pair_leaf_entries ()
- bool _has_differences (const std::vector< ublas_vec_type > &lhs, const std::vector< ublas_vec_type > &rhs) const
- bool _has_differences (const std::vector< ublas_vec_type > &lhs, const std::vector< ublas_vec_type > &rhs, std::vector< bool > &active)
- int _redist (ublas_vec_type &tmpv, subsum_vec_type &subsums, ublas_sym_matrix_type &dist_mat)
- void refine_cluster (cfentry_vec_type &entries)
- void _cluster (cfentry_vec_type &entries)

Private Attributes

- cfnode_sptr_type root
- cfnode_sptr_type leaf_dummy
- std::size_t mem_limit
- float_type dist_threshold
- dist_func_type dist_func
- dist_func_type absorb_dist_func
- std::size_t node_cnt

5.5.1 Detailed Description

template<boost::uint32_t dim>
class CFTree< dim >

class CFTree (clustering feature tree).

according to the paper, birch maintains btree-like data structure consisting of summarized clusters

Parameters

dim	dimensions of item, this parameter should be fixed before compiling

Definition at line 62 of file CFTree.h.

5.5.2 Member Typedef Documentation

5.5.2.1 template<boost::uint32_t dim> typedef std::pair<CFEntry<dim>*, CFEntry<dim>*> CFTree< dim >::cfentry_pair_type

Definition at line 87 of file CFTree.h.

5.5.2.2 template<boost::uint32_t dim> typedef std::vector<CFEntry<dim>*> CFTree< dim >::cfentry_ptr_vec_type

pair cfentry pointers.

Definition at line 88 of file CFTree.h.

5.5.2.3 template<boost::uint32_t dim> typedef std::vector<CFEntry<dim> > CFTree< dim >::cfentry_vec_type

distance function pointer.

Definition at line 90 of file CFTree.h.

5.5.2.4 template<boost::uint32_t dim> typedef boost::shared_ptr<CFNode<dim> > CFTree< dim >::cfnode_sptr_type

vector of items. pointer type of CFNode. shared_ptr is applied to preventing memory leakage. This node pointer is deleted, having no referencers

Definition at line 86 of file CFTree.h.

5.5.2.5 template<boost::uint32_t dim> typedef float_type(* CFTree< dim >::dist_func_type) (const CFEntry< dim > &, const CFEntry< dim > &)

vector of cfentry pointers.

Definition at line 89 of file CFTree.h.

5.5.2.6 template<boost::uint32_t dim> typedef float CFTree< dim >::float_type

enum for recognizing dimension outside this class.

Definition at line 80 of file CFTree.h.

5.5.2.7 template<boost::uint32_t dim> typedef std::vector<float_type> CFTree< dim >::item_vec_type

float type according to a precision - double, float, and so on.

Definition at line 81 of file CFTree.h.

5.5.2.8 template<boost::uint32_t dim> typedef std::vector< subcluster_summary > CFTree< dim >::subsum_vec_type

Definition at line 94 of file CFTree.h.

5.5.2.9 template<boost::uint32_t dim> typedef boost::numeric::ublas::symmetric_matrix<float_type> CFTree< dim >::ublas_sym_matrix_type

Definition at line 93 of file CFTree.h.

5.5.2.10 template<boost::uint32_t dim> typedef boost::numeric::ublas::vector<float_type> CFTree< dim >::ublas_vec_type

vector of cfentries.

Definition at line 92 of file CFTree.h.

5.5.3 Member Enumeration Documentation

5.5.3.1 template<boost::uint32_t dim> anonymous enum

Enumerator

fdim

Definition at line 78 of file CFTree.h.

```
78 { fdim = dim };
```

5.5.4 Constructor & Destructor Documentation

5.5.4.1 template<boost::uint32_t dim> CFTree< dim >::CFTree (float_type *in_dist_threshold*, std::size_t *in_mem_limit*, dist_func_type *in_dist_func* = _DistLarry, dist_func_type *in_absorb_dist_func* = _DistLarry) [inline]

CFTree construct with memory limit and designated distance functions

Parameters

in_dist_threshold	range within a CFEntry
in_mem_limit	memory limit to which CFTree can utilize, if CFTree overflows this limit, then distance threshold become larger to rebuild more compact CFTree
in_dist_func	distance function between CFEntries
in_dist_func	distance function tests if a new data-point should be absorbed or not

Definition at line 261 of file CFTree.h.

```
261
                                                        :
262         mem_limit(in_mem_limit), dist_threshold(in_dist_threshold),
      root( new CFNodeLeaf<dim>() ), dist_func(in_dist_func),
      absorb_dist_func(in_absorb_dist_func), node_cnt(1/* root node */),
263         leaf_dummy( new CFNodeLeaf<dim>() )
264     {
265         ((CFNodeLeaf<dim>*)leaf_dummy.get())->next =
      root;
266     }
```

5.5.4.2 template<boost::uint32_t dim> CFTree< dim >::~CFTree (void) [inline]

Definition at line 273 of file CFTree.h.

```
273 {}
```

5.5.5 Member Function Documentation

5.5.5.1 template<boost::uint32_t dim> void CFTree< dim >::_cluster (cfentry_vec_type & *entries*) [inline], [private]

Definition at line 1191 of file CFTree.h.

```
1192     {
1193         int n = (int) entries.size();
1194
1195         if (n <= 1)
1196             return;
1197
1198         if (dist_func == _DistD0 || dist_func ==
      _DistD1
1199             || dist_func == _DistLarry) {
1200             refine_cluster(entries);
1201         } else {
1202             HierarchicalClustering h(n - 1, dist_func);
1203             h.merge(entries);
1204             h.split(dist_threshold);
1205             h.result(entries);
1206         }
1207     }
```

5.5.5.2 template<boost::uint32_t dim> static float_type CFTree< dim >::_Diameter (const CFEntry< dim > & *e*) [inline], [static]

Definition at line 208 of file CFTree.h.

```
209     {
210         if (e.n <= 1)
211             return 0.0;
212
213         float_type temp = 0.0;
214         for (std::size_t i = 0; i < dim; i++)
215             temp += e.sum[i] / e.n * e.sum[i] / (e.n - 1);
216
217         float_type diameter = 2 * (e.sum_sq / (e.n - 1) - temp);
218
219         //assert(diameter >= 0.0);
220         return (std::max)(diameter, (float)0.0);
221     }
```

5.5.5.3 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD0 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Euclidean Distance of Centroid

Definition at line 99 of file CFTree.h.

```
100     {
101         float_type dist = 0.0;
102         float_type tmp;
103         for (std::size_t i = 0 ; i < dim ; i++) {
104             tmp =  lhs.sum[i]/lhs.n - rhs.sum[i]/rhs.n;
105             dist += tmp*tmp;
106         }
107         //assert(dist >= 0.0);
108         dist = sqrt(dist);
109         return (std::max)(dist, (float)0.0);
110     }
```

5.5.5.4 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD1 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Manhattan Distance of Centroid

Definition at line 113 of file CFTree.h.

```
114     {
115         float_type dist = 0.0;
116         float_type tmp;
117         for (std::size_t i = 0 ; i < dim ; i++) {
118             tmp = std::abs(lhs.sum[i]/lhs.n - rhs.sum[i]/rhs.n);
119             dist += tmp;
120         }
121         //assert(dist >= 0.0);
122         return (std::max)(dist, (float)0.0);
123     }
```

5.5.5.5 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD16 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Pairwise InterClusterDistance

Definition at line 180 of file CFTree.h.

```
181     {
182         float_type dist = 0.0;
183         float_type tmp;
184         for (std::size_t i = 0 ; i < dim ; i++) {
185             tmp =  lhs.sum[i]/lhs.n - rhs.sum[i]/rhs.n;
186             dist += pow(tmp, 16);
187         }
188         dist = pow(dist, 1.0/16);
189         //assert(dist >= 0.0);
190         return (std::max)(dist, (float)0.0);
191     }
```

5.5.5.6 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD2 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Pairwise IntraCluster Distance

Definition at line 126 of file CFTree.h.

```
127     {
128         float_type dot = 0.0;
129         for(std::size_t i = 0 ; i < dim ; i++)
130             dot += lhs.sum[i] * rhs.sum[i];
131 
132         float_type dist = ( rhs.n*lhs.sum_sq + lhs.n*rhs.
      sum_sq - 2*dot ) / (lhs.n*rhs.n);
133         //assert(dist >= 0.0);
134         return std::max(dist, (float)0.0);
135     }
```

5.5.5.7 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD3 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Pairwise InterClusterDistance

Definition at line 138 of file CFTree.h.

```
139     {
140         std::size_t tmpn = lhs.n + rhs.n;
141         float_type tmp1, tmp2 = 0.0;
142         for (std::size_t i = 0; i < dim; i++) {
143             tmp1 = lhs.sum[i] + rhs.sum[i];
144             tmp2 += tmp1 / tmpn * tmp1 / (tmpn - 1);
145         }
146         float_type dist = 2 * ((lhs.sum_sq + rhs.sum_sq) / (tmpn - 1) - tmp2);
147         //assert(dist >= 0.0);
148         return std::max(dist, (float)0.0);
149     }
```

5.5.5.8 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD4 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Pairwise InterClusterDistance

Definition at line 152 of file CFTree.h.

```
153     {
154         float_type dist = 0.0;
155         float_type tmp;
156         for (std::size_t i = 0 ; i < dim ; i++) {
157             tmp =  lhs.sum[i]/lhs.n - rhs.sum[i]/rhs.n;
158             dist += pow(tmp, 4);
159         }
160         dist = pow(dist, 1.0/4);
161         //assert(dist >= 0.0);
162         return (std::max)(dist, (float)0.0);
163     }
```

5.5.5.9 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistD8 (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Pairwise InterClusterDistance

Definition at line 166 of file CFTree.h.

```
167     {
168         float_type dist = 0.0;
169         float_type tmp;
170         for (std::size_t i = 0 ; i < dim ; i++) {
171             tmp =  lhs.sum[i]/lhs.n - rhs.sum[i]/rhs.n;
172             dist += pow(tmp, 8);
173         }
174         dist = pow(dist, 1.0/8);
175         //assert(dist >= 0.0);
176         return (std::max)(dist, (float)0.0);
177     }
```

5.5.5.10 template<boost::uint32_t dim> static float_type CFTree< dim >::_DistLarry (const CFEntry< dim > & *lhs*, const CFEntry< dim > & *rhs*) [inline], [static]

Definition at line 194 of file CFTree.h.

```
195     {
196         Eigen::VectorXf rLeft(dim), rRight(dim);
197         rLeft = VectorXf::Map(&(lhs.sum[0]), dim);
198         rRight = VectorXf::Map(&(rhs.sum[0]), dim);
199         rLeft /= lhs.n;
200         rRight /= rhs.n;
201 
202         //if(distanceMatrix)
203         //  return distanceMatrix[lhs.id][rhs.id];
204         //else
205             return getDisimilarity(rLeft,rRight,lhs.id,rhs.id,normOption,object);
206     }
```

5.5.5.11 template<boost::uint32_t dim> bool CFTree< dim >::_has_differences (const std::vector< ublas_vec_type > & *lhs*, const std::vector< ublas_vec_type > & *rhs*) const [inline], [private]

Definition at line 643 of file CFTree.h.

```
644     {
645         assert(lhs.size() == rhs.size());
646 
647         for (std::size_t i = 0; i < lhs.size(); i++) {
648             const ublas_vec_type diff = lhs[i] - rhs[i];
649             if (norm_2(diff) > std::numeric_limits<float_type>::epsilon())
650                 return true;
651         }
652         return false;
653     }
```

5.5.5.12 template<boost::uint32_t dim> bool CFTree< dim >::_has_differences (const std::vector< ublas_vec_type > & *lhs*, const std::vector< ublas_vec_type > & *rhs*, std::vector< bool > & *active*) [inline], [private]

Definition at line 655 of file CFTree.h.

```
656     {
657         assert( lhs.size() == rhs.size() );
658 
659         for (std::size_t i = 0; i < lhs.size(); i++){
660             const ublas_vec_type diff = lhs[i] - rhs[i];
661             active[i] = norm_2(diff) > std::numeric_limits<float_type>::epsilon();
662         }
663         return std::count( active.begin(), active.end(), true ) > 0;
664     }
```

5.5.5.13 template<boost::uint32_t dim> static float_type CFTree< dim >::_Radius (const CFEntry< dim > & *e*) [inline], [static]

Radius of the CFEntry

Definition at line 224 of file CFTree.h.

```
225     {
226         if (e.n <= 1)
227             return 0.0;
228 
229         float_type tmp0, tmp1 = 0.0;
230         for (std::size_t i = 0; i < dim; i++) {
231             tmp0 = e.sum[i] / e.n;
232             tmp1 += tmp0 * tmp0;
233         }
234         float_type radius = e.sum_sq / e.n - tmp1;
235 
236         //assert(radius >= 0.0);
237         return std::max(radius, (float)0.0);
238     }
```

5.5.5.14 template<boost::uint32_t dim> int CFTree< dim >::_redist (ublas_vec_type & *tmpv*, subsum_vec_type & *subsums*, ublas_sym_matrix_type & *dist_mat*) [inline], [private]

Definition at line 814 of file CFTree.h.

```
815     {
816         int imin, imax, i, k, n, start, end, median;
817         float d, tmpnorm, idist, tmpdist;
818         ublas_vec_type diff;
819 
820         i = 0;
821         n = (int) subsums.size();
822         tmpnorm = std::sqrt(inner_prod(tmpv, tmpv));
823 
824         // i=ClosestNorm(tmpnorm,norms,0,n-1);
825         // for efficiency, replace recursion by iteration
826         start = 0;
827         end = n - 1;
828         while (start < end) {
829             if (end - start == 1) {
830                 float_type norm_end = subsums[end].norm;
831                 float_type norm_start = subsums[start].norm;
832 
833                 i = tmpnorm > norm_end ? end : tmpnorm < norm_start ? start :
834                     tmpnorm - norm_start < norm_end - tmpnorm ? start : end;
835                 start = end = i;
836             } else {
837                 median = (start + end) / 2;
838                 float_type norm_med = subsums[median].norm;
839                 if (tmpnorm > norm_med)
840                     start = median;
841                 else
842                     end = median;
843             }
844         }
845 
846         diff = tmpv - subsums[i].center;
847         idist= inner_prod(diff, diff);
848 
849         // imin=MinLargerThan(tmpnorm-sqrt(idist),norms,0,n-1);
850         // imax=MaxSmallerThan(tmpnorm+sqrt(idist),norms,0,n-1);
851         // for efficiency, replace recursion by iteration
852 
853         tmpdist = tmpnorm - sqrt(idist);
854         start = 0;
855         end = n - 1;
856         while (start < end) {
857             median = (start + end) / 2;
858             float_type norm_med = subsums[median].norm;
859             if (tmpdist > norm_med)
860                 start = median + 1;
861             else
```

```
862                 end = median;
863         }
864         imin = start;
865 
866         tmpdist = tmpnorm + sqrt(idist);
867         start = 0;
868         end = n - 1;
869         while (start < end) {
870             median = (start + end + 1) / 2;
871             float_type norm_med = subsums[median].norm;
872             if (tmpdist < norm_med)
873                 end = median - 1;
874             else
875                 start = median;
876         }
877         imax = start;
878 
879         // ClosestCenter(i,idist,tmpv,centers,imin,imax,matrix,n);
880         // for efficiency, replace procedure by inline
881         k = imin;
882         while (k <= imax) {
883             if (dist_mat(k, i) <= 4 * idist) {
884                 diff = tmpv - subsums[k].center;
885                 d = inner_prod(diff, diff);
886                 if (d < idist) {
887                     idist = d;
888                     i = k;
889                 }
890             }
891             k++;
892         }
893         return i;
894     }
```

5.5.5.15 template<boost::uint32_t dim> float_type CFTree< dim >::average_dist_closest_pair_leaf_entries ( ) [inline], [private]

Definition at line 552 of file CFTree.h.

```
553     {
554         std::size_t total_n = 0;
555         float_type    total_d = 0.0;
556         float_type    dist;
557 
558         // determine new threshold
559         CFNodeLeaf<dim>* leaf = (CFNodeLeaf<dim>*)
    leaf_dummy.get();
560         while (leaf != NULL) {
561             if (leaf->size >= 2) {
562                 std::vector<float_type> min_dists(leaf->size, (std::numeric_limits<float_type>::max)())
    ;
563                 for (std::size_t i = 0; i < leaf->size - 1; i++) {
564                     for (std::size_t j = i + 1; j < leaf->size; j++) {
565                         dist = dist_func(leaf->entries[i], leaf->
    entries[j]);
566                         dist = dist >= 0.0 ? sqrt(dist) : 0.0;
567                         if (min_dists[i] > dist)
568                             min_dists[i] = dist;
569                         if (min_dists[j] > dist)
570                             min_dists[j] = dist;
571                     }
572                 }
573                 for (std::size_t i = 0; i < leaf->size; i++)
574                     total_d += min_dists[i];
575                 total_n += leaf->size;
576             }
577 
578             // next leaf
579             leaf = (CFNodeLeaf<dim>*) leaf->next.get();
580         }
581         return total_d / total_n;
582     }
```

5.5.5.16 template<boost::uint32_t dim> void CFTree< dim >::cluster (cfentry_vec_type & *entries*) [inline]

Definition at line 898 of file CFTree.h.

```
899     {
900         get_entries(entries);
901         _cluster(entries);
902     }
```

5.5.5.17 template<boost::uint32_t dim> bool CFTree< dim >::empty ( ) const [inline]

whether this CFTree is empty or not

Definition at line 276 of file CFTree.h.

```
276 { return root->IsEmpty(); }
```

5.5.5.18 template<boost::uint32_t dim> CFEntry<dim>* CFTree< dim >::find_close (CFNode< dim > * *node*, CFEntry< dim > & *new_entry*) [inline], [private]

Definition at line 386 of file CFTree.h.

```
387     {
388         CFEntry<dim>* begin = &node->entries[0];
389         CFEntry<dim>* end = begin + node->size;
390         CFEntry<dim>* e = std::min_element( begin, end, CloseEntryLessThan(new_entry,
    dist_func) );
391         return e != end ? e : NULL;
392     }
```

5.5.5.19 template<boost::uint32_t dim> void CFTree< dim >::find_farthest_pair (std::vector< CFEntry< dim > * > & *entries*, cfentry_pair_type & *far_pair*) [inline], [private]

Definition at line 532 of file CFTree.h.

```
533     {
534         //assert( entries.size() >= 2 );
535
536         float_type max_dist = -1.0;
537         for (std::size_t i = 0; i < entries.size() - 1; i++) {
538             for (std::size_t j = i + 1; j < entries.size(); j++) {
539                 CFEntry<dim>& e1 = *entries[i];
540                 CFEntry<dim>& e2 = *entries[j];
541
542                 float_type dist = dist_func(e1, e2);
543                 if (max_dist < dist) {
544                     max_dist = dist;
545                     far_pair.first = &e1;
546                     far_pair.second = &e2;
547                 }
548             }
549         }
550     }
```

5.5.5.20 template<boost::uint32_t dim> void CFTree< dim >::get_entries (cfentry_vec_type & *out_entries*) [inline]

get leaf entries

Definition at line 330 of file CFTree.h.

```
331     {
332         std::size_t n_leaf_entries = 0;
333         //leaf_iterator<dim> it = leaf_begin();
334         for( leaf_iterator<dim> it = leaf_begin() ; it !=
    leaf_end() ; ++it)
335             n_leaf_entries += it->size;
336 
337         out_entries.clear();
338         out_entries.reserve(n_leaf_entries);
339         for( leaf_iterator<dim> it = leaf_begin() ; it !=
    leaf_end() ; ++it )
340             std::copy( it->entries, it->entries + it->size, std::back_inserter(out_entries) );
341     }
```

5.5.5.21 template<boost::uint32_t dim> void CFTree< dim >::insert (item_vec_type & *item*) [inline]

inserting one data-point

Definition at line 279 of file CFTree.h.

```
280     {
281         if( item.size() != dim )
282             throw CFTreeInvalidItemSize();
283 
284         /*std::cout << "n is: " << CFEntry<dim>::n << std::endl;
285         for (int i = 0; i < dim; ++i)
286         {
287             std::cout << cfentry_vec_type[0].sum[i] << std::endl;
288         }*/
289 
290         insert(&item[0]);
291     }
```

5.5.5.22 template<boost::uint32_t dim> template<typename T > void CFTree< dim >::insert (T * *item*) [inline]

inserting one data-point with T typed

Definition at line 295 of file CFTree.h.

```
296     {
297         static std::size_t id = 0;
298         //CFEntry<dim> e(item);
299         //CFEntry<dim> e(item, id++);
300         CFEntry<dim> e(item, id%totalNodes);
301         ++id;
302         insert(e);
303         //if (id % 5448 == 0)
304         //std::cout << id << " item\n";
305     }
```

5.5.5.23 template<boost::uint32_t dim> void CFTree< dim >::insert (CFEntry< dim > & *e*) [inline]

inserting a new entry

Definition at line 308 of file CFTree.h.

```
309     {
310         bool bsplit;
311         insert(root.get(), e, bsplit);
312 
313         // there's no exception for the root as regard to splitting, indeed
314         if (bsplit) {
315             split_root(e);
316         }
317 
318         std::size_t curr_mem = node_cnt * sizeof(CFNode<dim> );
319         if (mem_limit > 0 && node_cnt * sizeof(CFNode<dim> ) >
    mem_limit) {
320             rebuild();
321         }
322     }
```

5.5.5.24 template<boost::uint32_t dim> void CFTree< dim >::insert (CFNode< dim > * *node*, CFEntry< dim > & *new_entry*, bool & *bsplit*) [inline], [private]

Definition at line 345 of file CFTree.h.

```
346     {
347         // empty node, it might be root node at first insertion
348         if (node->IsEmpty()) {
349             node->Add(new_entry);
350             bsplit = false;
351             return;
352         }
353 
354         CFEntry<dim>& close_entry = *find_close(node, new_entry);
355 
356         // non-leaf
357         if (close_entry.HasChild()) {
358             insert(close_entry.child.get(), new_entry, bsplit);
359 
360             // no more split
361             if (!bsplit)
362                 close_entry += (new_entry);
363             // split here
364             else
365                 split(*node, close_entry, new_entry, bsplit);
366         }
367         //leaf
368         else {
369             // absorb
370             if (absorb_dist_func(close_entry, new_entry) <
    dist_threshold) {
371                 close_entry += (new_entry);
372                 bsplit = false;
373             }
374             // add new_entry
375             else if (node->size < node->MaxEntrySize()) {
376                 node->Add(new_entry);
377                 bsplit = false;
378             }
379             // handle with the split cond. at parent-level
380             else {
381                 bsplit = true;
382             }
383         }
384     }
```

5.5.5.25 template<boost::uint32_t dim> leaf_iterator<dim> CFTree< dim >::leaf_begin ( ) [inline]

get the beginning of leaf iterators

Definition at line 325 of file CFTree.h.

```
325 { return leaf_iterator<dim>( (CFNodeLeaf<dim>*)((
      CFNodeLeaf<dim>*)leaf_dummy.get())->next.get()); }
```

5.5.5.26 template<boost::uint32_t dim> leaf_iterator<dim> CFTree< dim >::leaf_end ( ) [inline]

get the end of leaf iterators

Definition at line 327 of file CFTree.h.

```
327 { return leaf_iterator<dim>(NULL); }
```

5.5.5.27 template<boost::uint32_t dim> void CFTree< dim >::rearrange (cfentry_ptr_vec_type & *entries,* cfentry_pair_type & *far_pair,* CFEntry< dim > & *entry_lhs,* CFEntry< dim > & *entry_rhs*) [inline], [private]

Definition at line 511 of file CFTree.h.

```
512     {
513         entry_lhs.child->Add(*far_pair.first);
514         entry_lhs += *far_pair.first;
515         entry_rhs.child->Add(*far_pair.second);
516         entry_rhs += *far_pair.second;
517 
518         for (std::size_t i = 0; i < entries.size(); i++) {
519             CFEntry<dim>& e = *entries[i];
520             if (&e == far_pair.first || &e == far_pair.second)
521                 continue;
522 
523             float_type dist_first = dist_func(*far_pair.first, e);
524             float_type dist_second = dist_func(*far_pair.second, e);
525 
526             CFEntry<dim>& e_update = dist_first < dist_second ? entry_lhs : entry_rhs;
527             e_update.child->Add(e);
528             e_update += e;
529         }
530     }
```

5.5.5.28 template<boost::uint32_t dim> void CFTree< dim >::rebuild (bool *extend* = true) [inline]

rebuild tree from the existing leaf entries.

rebuilding cftree is regarded as clustering, because there could be overlapped cfentries. birch guarantees datapoints in cfentries within a range, but two data-points within a range can be separated to different cfentries

Parameters

extend	if true, the size of tree reaches to memory limit, so distance threshold enlarges. in case of both true and false, rebuilding CFTree from the existing leaves.

Definition at line 592 of file CFTree.h.

```
593     {
594         if( extend )
595         {
596             // decide the next threshold
597             float_type new_threshold = std::pow(
    average_dist_closest_pair_leaf_entries(),2);
598             dist_threshold = dist_threshold > new_threshold ?
    dist_threshold*2 : new_threshold;
599         }
600 
601         // construct a new tree by inserting all the node from the previous tree
602         CFTree<dim> new_tree( dist_threshold, mem_limit );
603 
604         CFNodeLeaf<dim>* leaf = (CFNodeLeaf<dim>*)
    leaf_dummy.get();
605         //std::cout << "leaf size is " << leaf->size << std::endl;
606         assert(leaf!=NULL);
607         while (leaf != NULL) {
608             for (std::size_t i = 0; i < leaf->size; i++)
609                 new_tree.insert(leaf->entries[i]);
610 
611             // next leaf
612             leaf = (CFNodeLeaf<dim>*) leaf->next.get();
613         }
614 
615         // really I'd like to replace the previous tree to the new one by
616         // stating " *this = new_tree; ", but it doesn't work because 'this' is const pointer
617         // copy root and dummy_node
618         // copy statistics variable
619 
620         root = new_tree.root;
621         leaf_dummy = new_tree.leaf_dummy;
622         node_cnt = new_tree.node_cnt;
623     }
```

5.5.5.29 template<boost::uint32_t dim> void CFTree< dim >::redist (std::vector< item_type< 4824u > >::iterator *begin*, std::vector< item_type< 4824u > >::iterator *end*, std::vector< CFEntry< 4824u > > & *entries*, std::vector< int > & *out_cid*) [inline]

Definition at line 774 of file CFTree.h.

```
775     {
776         using namespace boost::numeric::ublas;
777 
778         // prepare summaries for each subcluster
779         // summaries = ( center, radius, norm )
780         subsum_vec_type subclusters;
781         subclusters.reserve(entries.size());
782         for (std::size_t i = 0; i < entries.size(); i++) {
783             const CFEntry<dim>& e = entries[i];
784             ublas_vec_type center(dim);
785             std::copy(e.sum, e.sum + dim, center.begin());
786             center /= e.n;
787             subclusters.push_back(subcluster_summary(center,
    _Radius(e), std::sqrt(inner_prod(center, center))));
788         }
789 
790         std::sort(subclusters.begin(), subclusters.end(),
    subcluster_lessthan_norm());
791 
792         // in addition to an individual summary for each subcluster
793         // calculate pairwise euclidean distances of subclusters
794         std::size_t n = subclusters.size();
795         ublas_sym_matrix_type dist_mat(n, n);
796         for (std::size_t i = 0; i < n - 1; i++) {
797             for (std::size_t j = i + 1; j < n; j++) {
798                 ublas_vec_type diff = subclusters[i].center - subclusters[j].center;
799                 dist_mat(i, j) = inner_prod(diff, diff);
800             }
801         }
802 
803         out_cid.clear();
804         out_cid.reserve(end - begin);
805         for (std::vector<item_type<4824u> >::iterator it = begin; it != end; it++) {
806             ublas_vec_type v(dim);
807             std::copy(&(*it)[0], &(*it)[0] + dim, v.begin());
808             out_cid.push_back(_redist(v, subclusters, dist_mat));
809         }
810     }
```

5.5.5.30 template<boost::uint32_t dim> void CFTree< dim >::refine_cluster (cfentry_vec_type & *entries*) [inline], [private]

Definition at line 1116 of file CFTree.h.

```
1117     {
1118         std::vector<bool> merged(entries.size(), false);
1119 
1120         std::vector<std::size_t> not_visited;
1121         not_visited.reserve(entries.size());
1122 
1123         for (std::size_t i = 0; i < entries.size(); i++)
1124             not_visited.push_back(i);
1125 
1126         int temp = 0;
1127         while (!not_visited.empty()) {
1128             // pick any entry
1129             // std::cout << "iterations " << temp++ << std::endl;
1130             CFEntry<dim>& ref_entry = entries[not_visited.back()];
1131             CFEntry<dim> curr_entry = ref_entry;
1132             not_visited.pop_back();
1133 
1134             bool something_merged = false;
1135             for (std::size_t i = 0; i < not_visited.size(); i++) {
1136                 // index of next item
1137                 std::size_t v = not_visited[i];
1138                 CFEntry<dim>& e = entries[v];
1139                 if (dist_func(ref_entry, e) <= dist_threshold / 2) {
1140                     curr_entry += e;
1141                     merged[v] = true;
1142                     something_merged = true;
1143                 }
1144             }
1145 
1146             if (something_merged)
1147                 ref_entry = curr_entry;
1148 
1149             // remove if visited
1150             struct _remove_if_merged
1151             {
1152                 _remove_if_merged(std::vector<bool>& in_visited)
1153                         : visited(in_visited)
1154                 {
1155                 }
1156                 bool operator()(const std::size_t i) const
1157                 {
1158                     return visited[i];
1159                 }
1160             private:
1161                 std::vector<bool>& visited;
1162             };
1163 
1164             // prepare for next loop, removing visited indices
1165             if (something_merged)
1166                 not_visited.erase(std::remove_if(not_visited.begin(), not_visited.end(), _remove_if_merged(
    merged)),
1167                         not_visited.end());
1168         }
1169 
1170         struct _remove_if_merged_by_item
1171         {
1172             typedef CFEntry<dim> argument_type;
1173             _remove_if_merged_by_item(cfentry_vec_type& in_entries, std::vector<bool>&
    in_merged)
1174                     : entries(in_entries), merged(in_merged)
1175             {
1176             }
1177             bool operator()(const CFEntry<dim>& e) const
1178             {
1179                 std::ptrdiff_t i = (&e - &entries[0]);
1180                 return merged[i];
1181             }
1182 
1183         private:
1184             std::vector<bool>& merged;
1185             cfentry_vec_type& entries;
1186         };
1187         entries.erase(std::remove_if(entries.begin(), entries.end(), _remove_if_merged_by_item(entries,
    merged)),
1188                 entries.end());
1189     }
```

5.5.5.31 template<boost::uint32_t dim> void CFTree< dim >::setDistThreshold (const float_type & *in_dist_threshold*) [inline]

Definition at line 268 of file CFTree.h.

```
269     {
270         dist_threshold = in_dist_threshold;
271     }
```

5.5.5.32 template<boost::uint32_t dim> void CFTree< dim >::split (CFNode< dim > & *node*, CFEntry< dim > & *close_entry*, CFEntry< dim > & *new_entry*, bool & *bsplit*) [inline], [private]

Definition at line 394 of file CFTree.h.

```
395     {
396         CFNode<dim>* old_node = close_entry.child.get();
397         assert( old_node != NULL );
398 
399         // make the list of entries, old entries
400         cfentry_ptr_vec_type entries;
401         entries.reserve( old_node->MaxEntrySize() + 1 );
402         for( std::size_t i = 0 ; i < root->MaxEntrySize() ; i++ )
403             entries.push_back(&old_node->entries[i]);
404         entries.push_back(&new_entry);
405 
406         // find the farthest entry pair
407         cfentry_pair_type far_pair;
408         find_farthest_pair( entries, far_pair );
409 
410         bool node_is_leaf = old_node->IsLeaf();
411 
412         // make two split nodes
413         cfnode_sptr_type node_lhs( node_is_leaf ? (CFNode<dim>*) new
    CFNodeLeaf<dim>() : (CFNode<dim>*) new CFNodeItmd<dim>() );
414         cfnode_sptr_type node_rhs( node_is_leaf ? (CFNode<dim>*) new
    CFNodeLeaf<dim>() : (CFNode<dim>*) new CFNodeItmd<dim>() );
415 
416         // two entries for new root node
417         // and connect child node to the entries
418         CFEntry<dim> entry_lhs( node_lhs );
419         CFEntry<dim> entry_rhs( node_rhs );
420 
421         if( node_is_leaf )
422         {
423             assert( node_lhs->IsLeaf() && node_rhs->IsLeaf() );
424 
425             CFNodeLeaf<dim>* leaf_node = (CFNodeLeaf<dim>*)old_node;
426 
427             cfnode_sptr_type prev = leaf_node->prev;
428             cfnode_sptr_type next = leaf_node->next;
429 
430             if( prev != NULL )
431                 ((CFNodeLeaf<dim>*)prev.get())->next = node_lhs;
432             if( next != NULL )
433                 ((CFNodeLeaf<dim>*)next.get())->prev = node_rhs;
434 
435             ((CFNodeLeaf<dim>*)node_lhs.get())->prev = prev;
436             ((CFNodeLeaf<dim>*)node_lhs.get())->next = node_rhs;
437             ((CFNodeLeaf<dim>*)node_rhs.get())->prev = node_lhs;
438             ((CFNodeLeaf<dim>*)node_rhs.get())->next = next;
439         }
440 
441         // rearrange old entries to new entries
442         rearrange(entries, far_pair, entry_lhs, entry_rhs);
443 
444         // one old entry is divided into to new entries
445         // so the first one is included instead of old ones
446         node.Replace(close_entry, entry_lhs);
447 
448         // the full node indicates that this node have to be split as well
449         bsplit = node.IsFull();
450 
451         // copy the second entry newly created into return variable 'new_entry'
452         if( bsplit )
453             new_entry = entry_rhs;
```

```
454         // if affordable, not split, add the second entry to the node
455         else
456             node.Add(entry_rhs);
457 
458         // for statistics and mornitoring memory usage
459         node_cnt++;
460     }
```

5.5.5.33 template<boost::uint32_t dim> void CFTree< dim >::split_root (CFEntry< dim > & *e*) [inline], [private]

Definition at line 462 of file CFTree.h.

```
463     {
464         // make the list of entries, old entries
465         cfentry_ptr_vec_type entries;
466         entries.reserve(root->MaxEntrySize() + 1);
467         for( std::size_t i = 0 ; i < root->MaxEntrySize() ; i++ )
468             entries.push_back(&root->entries[i]);
469         entries.push_back(&e);
470 
471         // find the farthest entry pair
472         cfentry_pair_type far_pair;
473         find_farthest_pair( entries, far_pair );
474 
475         bool root_is_leaf = root->IsLeaf();
476 
477         // make two split nodes
478         cfnode_sptr_type node_lhs( root_is_leaf ? (CFNode<dim>*) new
    CFNodeLeaf<dim>() : (CFNode<dim>*) new CFNodeItmd<dim>() );
479         cfnode_sptr_type node_rhs( root_is_leaf ? (CFNode<dim>*) new
    CFNodeLeaf<dim>() : (CFNode<dim>*) new CFNodeItmd<dim>() );
480 
481         // two entries for new root node
482         // and connect child node to the entries
483         CFEntry<dim> entry_lhs( node_lhs );
484         CFEntry<dim> entry_rhs( node_rhs );
485 
486         // new root node result in two entries each of which has split node respectively
487         cfnode_sptr_type new_root( new CFNodeItmd<dim>() );
488 
489         // update prev/next links of newly created leaves
490         if( root_is_leaf )
491         {
492             assert( node_lhs->IsLeaf() && node_rhs->IsLeaf() );
493             ((CFNodeLeaf<dim>*)leaf_dummy.get())->next = node_lhs;
494             ((CFNodeLeaf<dim>*)node_lhs.get())->prev = leaf_dummy;
495             ((CFNodeLeaf<dim>*)node_lhs.get())->next = node_rhs;
496             ((CFNodeLeaf<dim>*)node_rhs.get())->prev = node_lhs;
497         }
498 
499         // rearrange old entries to new entries
500         rearrange( entries, far_pair, entry_lhs, entry_rhs );
501 
502         // substitute new_root to 'root' variable
503         new_root->Add(entry_lhs);
504         new_root->Add(entry_rhs);
505         root = new_root;
506 
507         // for statistics and mornitoring memory usage
508         node_cnt++;
509     }
```

5.5.6 Member Data Documentation

5.5.6.1 template<boost::uint32_t dim> dist_func_type CFTree< dim >::absorb_dist_func [private]

Definition at line 635 of file CFTree.h.

5.5.6.2 template<boost::uint32_t dim> dist_func_type CFTree< dim >::dist_func [private]

Definition at line 634 of file CFTree.h.

5.5.6.3 template<boost::uint32_t dim> float_type CFTree< dim >::dist_threshold [private]

Definition at line 633 of file CFTree.h.

5.5.6.4 template<boost::uint32_t dim> cfnode_sptr_type CFTree< dim >::leaf_dummy [private]

Definition at line 629 of file CFTree.h.

5.5.6.5 template<boost::uint32_t dim> std::size_t CFTree< dim >::mem_limit [private]

Definition at line 632 of file CFTree.h.

5.5.6.6 template<boost::uint32_t dim> std::size_t CFTree< dim >::node_cnt [private]

Definition at line 638 of file CFTree.h.

5.5.6.7 template<boost::uint32_t dim> int CFTree< dim >::normOption = -1 [static]

Definition at line 72 of file CFTree.h.

5.5.6.8 template<boost::uint32_t dim> MetricPreparation CFTree< dim >::object = MetricPreparation() [static]

Definition at line 71 of file CFTree.h.

5.5.6.9 template<boost::uint32_t dim> cfnode_sptr_type CFTree< dim >::root [private]

Definition at line 628 of file CFTree.h.

5.5.6.10 template<boost::uint32_t dim> int CFTree< dim >::totalNodes = 0 [static]

Definition at line 73 of file CFTree.h.

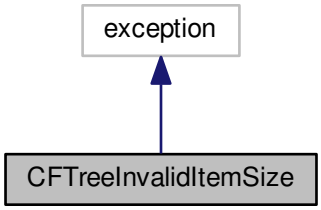
The documentation for this class was generated from the following files:

- CFTree.h
- ClusterAnalysis.h

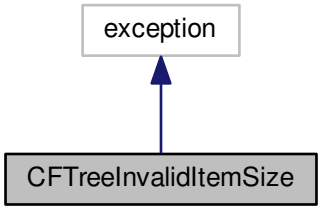
5.6 CFTreeInvalidItemSize Struct Reference

`#include <CFEntry.h>`

Inheritance diagram for CFTreeInvalidItemSize:



Collaboration diagram for CFTreeInvalidItemSize:



5.6.1 Detailed Description

this exception is produced when the current item size is not suitable.

Definition at line 27 of file CFEntry.h.

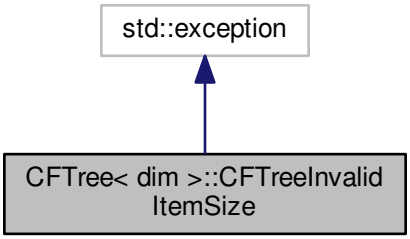
The documentation for this struct was generated from the following file:

- CFEntry.h

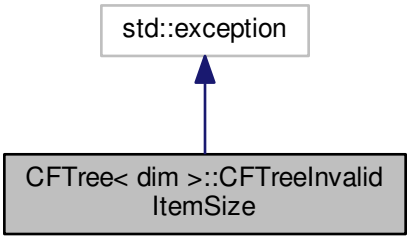
5.7 CFTree< dim >::CFTreeInvalidItemSize Struct Reference

`#include <CFTree.h>`

Inheritance diagram for CFTree< dim >::CFTreeInvalidItemSize:



Collaboration diagram for CFTree< dim >::CFTreeInvalidItemSize:



5.7.1 Detailed Description

template<boost::uint32_t dim>
struct CFTree< dim >::CFTreeInvalidItemSize

this exception is produced when the current item size is not suitable.

Definition at line 76 of file CFTree.h.

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

- CFTree.h

5.8 CFTree< dim >::CloseEntryLessThan Struct Reference

Public Member Functions

- CloseEntryLessThan (const CFEntry< dim > &in_base_entry, const dist_func_type &in_dist_func)
- bool operator() (const CFEntry< dim > &lhs, const CFEntry< dim > &rhs)

Public Attributes

- const CFEntry< dim > & base_entry
- const dist_func_type & dist_func

5.8.1 Detailed Description

template<boost::uint32_t dim>
struct CFTree< dim >::CloseEntryLessThan

Functor is used for choosing the closest one

Definition at line 242 of file CFTree.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 template<boost::uint32_t dim> CFTree< dim >::CloseEntryLessThan::CloseEntryLessThan (const CFEntry< dim > & *in_base_entry,* const dist_func_type & *in_dist_func*) [inline]

Definition at line 244 of file CFTree.h.

```
244 : base_entry(in_base_entry), dist_func(in_dist_func) {}
```

5.8.3 Member Function Documentation

5.8.3.1 template<boost::uint32_t dim> bool CFTree< dim >::CloseEntryLessThan::operator() (const CFEntry< dim > & *lhs,* const CFEntry< dim > & *rhs*) [inline]

Definition at line 245 of file CFTree.h.

```
245 { return dist_func(lhs, base_entry) < dist_func(rhs,
      base_entry); }
```

5.8.4 Member Data Documentation

5.8.4.1 template<boost::uint32_t dim> const CFEntry<dim>& CFTree< dim >::CloseEntryLessThan::base_entry

Definition at line 247 of file CFTree.h.

5.8.4.2 template<boost::uint32_t dim> const dist_func_type& CFTree< dim >::CloseEntryLessThan::dist_func

Definition at line 248 of file CFTree.h.

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

- CFTree.h

5.9 FileIndex Struct Reference

`#include <ClusterAnalysis.h>`

Public Member Functions

- FileIndex ()
- ~FileIndex ()

Public Attributes

- int vertexCount
- int maxElement

5.9.1 Detailed Description

Definition at line 78 of file ClusterAnalysis.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 FileIndex::FileIndex ( ) [inline]

Definition at line 81 of file ClusterAnalysis.h.

```
82     {}
```

5.9.2.2 FileIndex::~FileIndex ( ) [inline]

Definition at line 84 of file ClusterAnalysis.h.

```
85     {}
```

5.9.3 Member Data Documentation

5.9.3.1 int FileIndex::maxElement

Definition at line 80 of file ClusterAnalysis.h.

5.9.3.2 int FileIndex::vertexCount

Definition at line 80 of file ClusterAnalysis.h.

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

- ClusterAnalysis.h

5.10 CFTree< dim >::HierarchicalClustering Struct Reference

Public Types

- typedef boost::numeric::ublas::symmetric_matrix< float_type > dist_matrix_type

Public Member Functions

- HierarchicalClustering (int n, dist_func_type &in_dist_func)
- void merge (cfentry_vec_type &entries)
- void split (float_type ft)
- short _split (float_type ft)
- void result (cfentry_vec_type &entries)
- int farthest_merge (int chainptr)
- int largest_merge (int chainptr, float_type dist_threshold)
- int nearest_neighbor (int CurI, int n, int *checked, dist_matrix_type &dist)
- int pick_one_unchecked (int n, int *checked)
- void update_distance (int n1, int n2, int CurI, int NextI, int n, int *checked, dist_matrix_type &dist)

Public Attributes

- int size
- int step
- std::vector< int > ii
- std::vector< int > jj
- std::vector< CFEntry< dim > > cf
- std::vector< float_type > dd
- std::vector< int > chain
- int chainptr
- short stopchain
- dist_func_type & dist_func

5.10.1 Detailed Description

template<boost::uint32_t dim>
struct CFTree< dim >::HierarchicalClustering

Definition at line 906 of file CFTree.h.

5.10.2 Member Typedef Documentation

5.10.2.1 template<boost::uint32_t dim> typedef boost::numeric::ublas::symmetric_matrix<float_type> CFTree< dim >::HierarchicalClustering::dist_matrix_type

Definition at line 908 of file CFTree.h.

5.10.3 Constructor & Destructor Documentation

5.10.3.1 template<boost::uint32_t dim> CFTree< dim >::HierarchicalClustering::HierarchicalClustering (int *n*, dist_func_type & *in_dist_func*) [inline]

Definition at line 910 of file CFTree.h.

```
911                 : size(n), step(-1), ii(n), jj(n), cf(n), dd(n),
      dist_func(in_dist_func), chain(n + 1), chainptr(-1),
      stopchain(FALSE)
912         {
913         }
```

5.10.4 Member Function Documentation

5.10.4.1 template<boost::uint32_t dim> short CFTree< dim >::HierarchicalClustering::_split (float_type *ft*) [inline]

Definition at line 1001 of file CFTree.h.

```
1002         {
1003             int i, j;
1004
1005             if (chainptr == size)
1006                 return FALSE;
1007
1008             i = largest_merge(chainptr, ft);
1009             if (i != -1) {
1010                 j = -chain[i] - 1;
1011                 chain[i] = ii[j];
1012                 chain[++chainptr] = jj[j];
1013                 return TRUE;
1014             }
1015
1016             stopchain = TRUE;
1017             return FALSE;
1018         }
```

5.10.4.2 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::farthest_merge (int *chainptr*) [inline]

Definition at line 1031 of file CFTree.h.

```
1032         {
1033             if (chainptr <= 0)
1034                 return chainptr;
1035 
1036             float d, dmax = 0;
1037             int i, imax = -1;
1038             for (i = 0; i <= chainptr; i++) {
1039                 if (chain[i] < 0) {
1040                     d = dd[-chain[i] - 1];
1041                     if (d > dmax) {
1042                         imax = i;
1043                         dmax = d;
1044                     }
1045                 }
1046             }
1047             return imax;
1048         }
```

5.10.4.3 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::largest_merge (int *chainptr*, float_type *dist_threshold*) [inline]

Definition at line 1051 of file CFTree.h.

```
1052         {
1053             for (int i = 0; i <= chainptr; i++) {
1054                 if (chain[i] < 0) {
1055                     if (_Diameter(cf[-chain[i] - 1]) >
     dist_threshold)
1056                         return i;
1057                 }
1058             }
1059             return -1;
1060         }
```

5.10.4.4 template<boost::uint32_t dim> void CFTree< dim >::HierarchicalClustering::merge (cfentry_vec_type & *entries*) [inline]

Definition at line 915 of file CFTree.h.

```
916         {
917             int nentry = (int) entries.size();
918             int i, j, n1, n2;
919 
920             int CurI, PrevI, NextI;
921             int uncheckcnt = nentry;
922 
923             std::vector<int> checked(nentry);
924             for (i = 0; i < nentry; i++)
925                 checked[i] = i + 1;
926             // 0: invalid after being merged to other entries
927             // positive 1..nentry+1 :     original entries
928             // negative -1..-(nentry-1) : merged entries
929 
930             // get initial distances
931             // std::vector<float_type> dist(nentry*(nentry-1)/2);
932             dist_matrix_type dist(nentry, nentry);
933 
934             for (i = 0; i < nentry - 1; i++)
935                 for (j = i + 1; j < nentry; j++)
936                     dist(i, j) = dist_func(entries[i], entries[j]);
937 
938             CurI = rand() % nentry;          // step1
939             chain[++chainptr] = CurI;
```

```
940
941             while (uncheckcnt > 1) {
942                 // step4
943                 if (chainptr == -1) {
944                     chainptr++;
945                     chain[chainptr] = pick_one_unchecked(nentry, &checked[0]
    );
946                 }
947                 PrevI = chainptr > 0 ? chain[chainptr - 1] : -1;
948                 stopchain = FALSE;
949
950                 // step2
951                 while (stopchain == FALSE) {
952                     CurI = chain[chainptr];
953                     NextI = nearest_neighbor(CurI, nentry, &checked[0], dist);
954
955                     // it is impossible NextI be -1 because uncheckcnt>1
956                     if (NextI == PrevI)
957                         stopchain = TRUE;
958                     else {
959                         chain[++chainptr] = NextI;
960                         PrevI = CurI;
961                     }
962                 } // end of while for step 2
963
964                 step++;
965
966                 // step3
967                 ii[step] = checked[CurI];
968                 jj[step] = checked[NextI];
969
970                 dd[step] = dist(CurI, NextI);
971
972                 bool curr_org = checked[CurI] > 0;
973                 bool next_org = checked[NextI] > 0;
974
975                 CFEntry<dim>& curr_entry = curr_org ? entries[CurI] :
    cf[-checked[CurI] - 1];
976                 CFEntry<dim>& next_entry = next_org ? entries[NextI] :
    cf[-checked[NextI] - 1];
977                 n1 = curr_entry.n;
978                 n2 = next_entry.n;
979                 cf[step] = curr_entry + next_entry;
980
981                 update_distance(n1, n2, CurI, NextI, nentry, &checked[0], dist);
982                 uncheckcnt--;
983                 checked[CurI] = -(step + 1);
984                 checked[NextI] = 0;
985                 chainptr--;
986                 chainptr--;
987             } //end of while (uncheckcnt>1)
988
989             // prepare for SplitHierarchy
990             stopchain = FALSE;
991             chainptr = 0;
992             chain[chainptr] = -(step + 1);
993         }
```

5.10.4.5 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::nearest_neighbor (int *CurI*, int *n*, int * *checked*, dist_matrix_type & *dist*) [inline]

Definition at line 1063 of file CFTree.h.

```
1064         {
1065             int imin = 0;
1066             float d, dmin = (std::numeric_limits<float_type>::max)();
1067             for (int i = 0; i < n; i++) {
1068                 if (i == CurI || checked[i] == 0)
1069                     continue;
1070
1071                 d = dist(i, CurI);
1072                 if (d < dmin) {
1073                     dmin = d;
1074                     imin = i;
1075                 }
1076             }
1077
1078             return dmin < (std::numeric_limits<float_type>::max)() ? imin : -1;
1079         }
```

5.10.4.6 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::pick_one_unchecked (int *n*, int * *checked*) [inline]

Definition at line 1082 of file CFTree.h.

```
1083         {
1084             int i, j = rand() % n;
1085             for (i = 0; i < n; i++)
1086                 if (checked[(i + j) % n] != 0)
1087                     return (i + j) % n;
1088             return -1;
1089         }
```

5.10.4.7 template<boost::uint32_t dim> void CFTree< dim >::HierarchicalClustering::result (cfentry_vec_type & *entries*) [inline]

Definition at line 1020 of file CFTree.h.

```
1021         {
1022             int j;
1023             std::vector<CFEntry<dim> > tmpentries(chainptr + 1);
1024             for (j = 0; j <= chainptr; j++) {
1025                 tmpentries[j] = chain[j] < 0 ? cf[-chain[j] - 1] : entries[
     chain[j] - 1];
1026             }
1027             entries = tmpentries;
1028         }
```

5.10.4.8 template<boost::uint32_t dim> void CFTree< dim >::HierarchicalClustering::split (float_type *ft*) [inline]

Definition at line 995 of file CFTree.h.

```
996         {
997             while (_split(ft))
998                 /* nothing to do */;
999         }
```

5.10.4.9 template<boost::uint32_t dim> void CFTree< dim >::HierarchicalClustering::update_distance (int *n1*, int *n2*, int *CurI*, int *NextI*, int *n*, int * *checked*, dist_matrix_type & *dist*) [inline]

Definition at line 1092 of file CFTree.h.

```
1093         {
1094             for (int i = 0; i < n; i++) {
1095                 if (i == CurI || i == NextI)
1096                     continue;
1097
1098                 if (checked[i] != 0)
1099                     dist(i, CurI) = (n1 * dist(i, CurI) + n2 * dist(i, NextI)) / (n1 + n2);
1100             }
1101         }
```

5.10.5 Member Data Documentation

5.10.5.1 template<boost::uint32_t dim> std::vector<CFEntry<dim> > CFTree< dim >::HierarchicalClustering::cf

Definition at line 1107 of file CFTree.h.

5.10.5.2 template<boost::uint32_t dim> std::vector<int> CFTree< dim >::HierarchicalClustering::chain

Definition at line 1110 of file CFTree.h.

5.10.5.3 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::chainptr

Definition at line 1111 of file CFTree.h.

5.10.5.4 template<boost::uint32_t dim> std::vector<float_type> CFTree< dim >::HierarchicalClustering::dd

Definition at line 1108 of file CFTree.h.

5.10.5.5 template<boost::uint32_t dim> dist_func_type& CFTree< dim >::HierarchicalClustering::dist_func

Definition at line 1113 of file CFTree.h.

5.10.5.6 template<boost::uint32_t dim> std::vector<int> CFTree< dim >::HierarchicalClustering::ii

Definition at line 1105 of file CFTree.h.

5.10.5.7 template<boost::uint32_t dim> std::vector<int> CFTree< dim >::HierarchicalClustering::jj

Definition at line 1106 of file CFTree.h.

5.10.5.8 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::size

Definition at line 1103 of file CFTree.h.

5.10.5.9 template<boost::uint32_t dim> int CFTree< dim >::HierarchicalClustering::step

Definition at line 1104 of file CFTree.h.

5.10.5.10 template<boost::uint32_t dim> short CFTree< dim >::HierarchicalClustering::stopchain

Definition at line 1112 of file CFTree.h.

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

- CFTree.h

5.11 HierarchicalClustering Struct Reference

Public Types

- typedef boost::numeric::ublas::symmetric_matrix< float_type > dist_matrix_type

Public Member Functions

- HierarchicalClustering (int n, dist_func_type &in_dist_func)
- void merge (cfentry_vec_type &entries)
- void split (float_type ft)
- short _split (float_type ft)
- void result (cfentry_vec_type &entries)
- int farthest_merge (int chainptr)
- int largest_merge (int chainptr, float_type dist_threshold)
- int nearest_neighbor (int CurI, int n, int *checked, dist_matrix_type &dist)
- int pick_one_unchecked (int n, int *checked)
- void update_distance (int n1, int n2, int CurI, int NextI, int n, int *checked, dist_matrix_type &dist)

Public Attributes

- int size
- int step
- std::vector< int > ii
- std::vector< int > jj
- std::vector< CFEntry > cf
- std::vector< float_type > dd
- std::vector< int > chain
- int chainptr
- short stopchain
- dist_func_type & dist_func

5.11.1 Detailed Description

Definition at line 40 of file CFTree_CFCluster.h.

5.11.2 Member Typedef Documentation

5.11.2.1 typedef boost::numeric::ublas::symmetric_matrix<float_type> HierarchicalClustering::dist_matrix_type

Definition at line 42 of file CFTree_CFCluster.h.

5.11.3 Constructor & Destructor Documentation

5.11.3.1 HierarchicalClustering::HierarchicalClustering (int *n,* dist_func_type & *in_dist_func*) [inline]

Definition at line 44 of file CFTree_CFCluster.h.

```
44                                                                          : size(n),
      step(-1), ii(n), jj(n), cf(n), dd(n), dist_func(in_dist_func),
      chain(n+1), chainptr(-1), stopchain(FALSE)
45              {
46              }
```

5.11.4 Member Function Documentation

5.11.4.1 short HierarchicalClustering::_split (float_type *ft*) [inline]

Definition at line 138 of file CFTree_CFCluster.h.

```
139             {
140                 int i, j;
141
142                 if ( chainptr == size )
143                     return FALSE;
144
145                 i = largest_merge(chainptr, ft);
146                 if (i!=-1)
147                 {
148                     j = -chain[i]-1;
149                     chain[i] = ii[j];
150                     chain[++chainptr]=jj[j];
151                     return TRUE;
152                 }
153
154                 stopchain = TRUE;
155                 return FALSE;
156             }
```

5.11.4.2 int HierarchicalClustering::farthest_merge (int *chainptr*) [inline]

Definition at line 170 of file CFTree_CFCluster.h.

```
171             {
172                 if (chainptr<=0)
173                     return chainptr;
174
175                 float d, dmax = 0;
176                 int    i, imax = -1;
177                 for (i=0; i<=chainptr; i++)
178                 {
179                     if (chain[i]<0)
180                     {
181                         d = dd[-chain[i]-1];
182                         if (d>dmax) {imax = i; dmax = d;}
183                     }
184                 }
185                 return imax;
186             }
```

5.11.4.3 int HierarchicalClustering::largest_merge (int *chainptr,* float_type *dist_threshold*) [inline]

Definition at line 189 of file CFTree_CFCluster.h.

```
190             {
191                 for (int i=0; i<=chainptr; i++)
192                 {
193                     if (chain[i]<0)
194                     {
195                         if ( _Diameter(cf[-chain[i]-1]) > dist_threshold)
196                             return i;
197                     }
198                 }
199                 return -1;
200             }
```

5.11.4.4 void HierarchicalClustering::merge (cfentry_vec_type & *entries*) [inline]

Definition at line 48 of file CFTree_CFCluster.h.

```
49             {
50                 int     nentry = (int)entries.size();
51                 int     i,j,n1,n2;
52 
53                 int     CurI, PrevI, NextI;
54                 int     uncheckcnt = nentry;
55 
56                 std::vector<int> checked(nentry);
57                 for (i=0;i<nentry;i++)
58                     checked[i]=i+1;
59                 // 0: invalid after being merged to other entries
60                 // positive 1..nentry+1 :     original entries
61                 // negative -1..-(nentry-1) : merged entries
62 
63                 // get initial distances
64                 // std::vector<float_type> dist(nentry*(nentry-1)/2);
65                 dist_matrix_type dist(nentry, nentry);
66 
67                 for (i=0; i<nentry-1; i++)
68                     for (j=i+1; j<nentry; j++)
69                         dist(i, j) = dist_func(entries[i],entries[j]);
70 
71                 CurI = rand() % nentry;         // step1
72                 chain[++chainptr]=CurI;
73 
74                 while (uncheckcnt>1)
75                 {
76                     // step4
77                     if (chainptr==-1)
78                     {
79                         chainptr++;
80                         chain[chainptr]=pick_one_unchecked(nentry, &checked[
    0]);
81                     }
82                     PrevI = chainptr > 0 ? chain[chainptr-1] : -1;
83                     stopchain=FALSE;
84 
85                     // step2
86                     while (stopchain==FALSE)
87                     {
88                         CurI=chain[chainptr];
89                         NextI = nearest_neighbor(CurI,nentry,&checked[0], dist);
90 
91                         // it is impossible NextI be -1 because uncheckcnt>1
92                         if (NextI==PrevI)
93                             stopchain = TRUE;
94                         else
95                         {
96                             chain[++chainptr]=NextI;
97                             PrevI = CurI;
98                         }
99                     } // end of while for step 2
100 
101                     step++;
102 
103                     // step3
```

```
104                     ii[step] = checked[CurI];
105                     jj[step] = checked[NextI];
106 
107                     dd[step] = dist(CurI, NextI);
108 
109                     bool curr_org = checked[CurI] > 0;
110                     bool next_org = checked[NextI] > 0;
111 
112                     CFEntry& curr_entry = curr_org ? entries[CurI] : cf[-checked[CurI]-1];
113                     CFEntry& next_entry = next_org ? entries[NextI] : cf[-checked[NextI]-1];
114                     n1 = curr_entry.n;
115                     n2 = next_entry.n;
116                     cf[step] = curr_entry + next_entry;
117 
118                     update_distance(n1,n2,CurI,NextI,nentry,&checked[0], dist);
119                     uncheckcnt--;
120                     checked[CurI] = -(step+1);
121                     checked[NextI] = 0;
122                     chainptr--;
123                     chainptr--;
124                 } //end of while (uncheckcnt>1)
125 
126                 // prepare for SplitHierarchy
127                 stopchain = FALSE;
128                 chainptr = 0;
129                 chain[chainptr] = -(step+1);
130             }
```

5.11.4.5 int HierarchicalClustering::nearest_neighbor (int *CurI*, int *n*, int * *checked*, dist_matrix_type & *dist*) [inline]

Definition at line 203 of file CFTree_CFCluster.h.

```
204             {
205                 int    imin=0;
206                 float d, dmin = (std::numeric_limits<float_type>::max)();
207                 for( int i = 0 ; i < n ; i++ )
208                 {
209                     if( i == CurI || checked[i] == 0 )
210                         continue;
211 
212                     d = dist(i, CurI);
213                     if (d < dmin)
214                     {
215                         dmin=d;
216                         imin=i;
217                     }
218                 }
219 
220                 return dmin < (std::numeric_limits<float_type>::max)() ? imin : -1;
221             }
```

5.11.4.6 int HierarchicalClustering::pick_one_unchecked (int *n*, int * *checked*) [inline]

Definition at line 224 of file CFTree_CFCluster.h.

```
225             {
226                 int i,j = rand() % n;
227                 for (i=0;i<n;i++)
228                     if (checked[(i+j)%n]!=0)
229                         return (i+j)%n;
230                 return -1;
231             }
```

5.11.4.7 void HierarchicalClustering::result (cfentry_vec_type & *entries*) [inline]

Definition at line 158 of file CFTree_CFCluster.h.

```
159             {
160                 int j;
161                 std::vector<CFEntry> tmpentries( chainptr + 1 );
162                 for( j = 0 ; j <= chainptr ; j++ )
163                 {
164                     tmpentries[j] = chain[j] < 0 ? cf[-chain[j]-1] : entries[
    chain[j]-1];
165                 }
166                 entries = tmpentries;
167             }
```

5.11.4.8 void HierarchicalClustering::split (float_type *ft*) [inline]

Definition at line 132 of file CFTree_CFCluster.h.

```
133             {
134                 while( _split(ft) )
135                     /* nothing to do */;
136             }
```

5.11.4.9 void HierarchicalClustering::update_distance (int *n1*, int *n2*, int *Curl*, int *Nextl*, int *n*, int * *checked*, dist_matrix_type & *dist*) [inline]

Definition at line 234 of file CFTree_CFCluster.h.

```
235             {
236                 for( int i = 0 ; i < n ; i++ )
237                 {
238                     if( i == CurI || i == NextI )
239                         continue;
240 
241                     if( checked[i] != 0 )
242                         dist(i, CurI) = (n1 * dist(i, CurI) + n2 * dist(i, NextI)) / (n1 + n2);
243                 }
244             }
```

5.11.5 Member Data Documentation

5.11.5.1 std::vector<CFEntry> HierarchicalClustering::cf

Definition at line 250 of file CFTree_CFCluster.h.

5.11.5.2 std::vector<int> HierarchicalClustering::chain

Definition at line 253 of file CFTree_CFCluster.h.

5.11.5.3 int HierarchicalClustering::chainptr

Definition at line 254 of file CFTree_CFCluster.h.

5.11.5.4 std::vector<float_type> HierarchicalClustering::dd

Definition at line 251 of file CFTree_CFCluster.h.

5.11.5.5 dist_func_type& HierarchicalClustering::dist_func

Definition at line 256 of file CFTree_CFCluster.h.

5.11.5.6 std::vector<int> HierarchicalClustering::ii

Definition at line 248 of file CFTree_CFCluster.h.

5.11.5.7 std::vector<int> HierarchicalClustering::jj

Definition at line 249 of file CFTree_CFCluster.h.

5.11.5.8 int HierarchicalClustering::size

Definition at line 246 of file CFTree_CFCluster.h.

5.11.5.9 int HierarchicalClustering::step

Definition at line 247 of file CFTree_CFCluster.h.

5.11.5.10 short HierarchicalClustering::stopchain

Definition at line 255 of file CFTree_CFCluster.h.

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

- CFTree_CFCluster.h

5.12 item_type< dim > Struct Template Reference

`#include <item_type.h>`

Public Member Functions

- item_type ()
- item_type (float *in_item)
- float & operator[ ] (int i)
- float operator[ ] (int i) const
- std::size_t size () const
- int & cid ()
- const int cid () const

Public Attributes

- float item [dim]
- int id

5.12.1 Detailed Description

template<boost::uint32_t dim>
struct item_type< dim >

Definition at line 14 of file item_type.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 template<boost::uint32_t dim> item_type< dim >::item_type ( ) [inline]

Definition at line 16 of file item_type.h.

```
16 : id(0) { std::fill( item, item + sizeof(item)/sizeof(item[0]), 0 ); }
```

5.12.2.2 template<boost::uint32_t dim> item_type< dim >::item_type (float * *in_item*) [inline]

Definition at line 17 of file item_type.h.

```
17 : id(0) { std::copy(in_item, in_item+sizeof(item)/sizeof(item[0]),
      item); }
```

5.12.3 Member Function Documentation

5.12.3.1 template<boost::uint32_t dim> int& item_type< dim >::cid ( ) [inline]

Definition at line 22 of file item_type.h.

```
22 { return id; }
```

5.12.3.2 template<boost::uint32_t dim> const int item_type< dim >::cid ( ) const [inline]

Definition at line 23 of file item_type.h.

```
23 { return id; }
```

5.12.3.3 template<boost::uint32_t dim> float& item_type< dim >::operator[ ] (int *i*) [inline]

Definition at line 18 of file item_type.h.

```
18 { return item[i]; }
```

5.12.3.4 template<boost::uint32_t dim> float item_type< dim >::operator[ ] (int *i*) const [inline]

Definition at line 19 of file item_type.h.

```
19 { return item[i]; }
```

5.12.3.5 template<boost::uint32_t dim> std::size_t item_type< dim >::size ( ) const [inline]

Definition at line 20 of file item_type.h.

```
20 { return sizeof(item)/sizeof(item[0]); }
```

5.12.4 Member Data Documentation

5.12.4.1 template<boost::uint32_t dim> int item_type< dim >::id

Definition at line 27 of file item_type.h.

5.12.4.2 template<boost::uint32_t dim> float item_type< dim >::item[dim]

Definition at line 26 of file item_type.h.

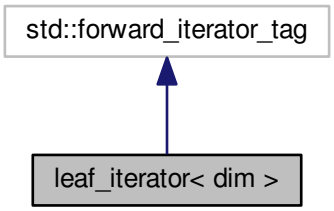
The documentation for this struct was generated from the following file:

- item_type.h

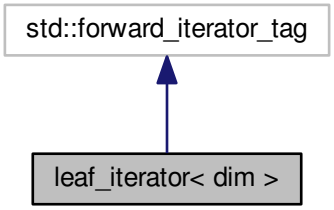
5.13 leaf_iterator< dim > Struct Template Reference

`#include <leaf_iterator.h>`

Inheritance diagram for leaf_iterator< dim >:



Collaboration diagram for leaf_iterator< dim >:



Public Member Functions

- leaf_iterator (CFNodeLeaf< dim > *in_leaf)
- leaf_iterator operator++ ()
- bool operator!= (const leaf_iterator rhs) const
- CFNodeLeaf< dim > & operator* ()
- CFNodeLeaf< dim > * operator-> ()

Public Attributes

- CFNodeLeaf< dim > * leaf

5.13.1 Detailed Description

template<boost::uint32_t dim>
struct leaf_iterator< dim >

leaf iterator

Definition at line 21 of file leaf_iterator.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 template<boost::uint32_t dim> leaf_iterator< dim >::leaf_iterator (CFNodeLeaf< dim > * *in_leaf*) [inline]

Definition at line 23 of file leaf_iterator.h.

```
23 : leaf( in_leaf ) {}
```

5.13.3 Member Function Documentation

5.13.3.1 template<boost::uint32_t dim> bool leaf_iterator< dim >::operator!= (const leaf_iterator< dim > *rhs*) const [inline]

Definition at line 25 of file leaf_iterator.h.

```
25 { return !(leaf == rhs.leaf); }
```

5.13.3.2 template<boost::uint32_t dim> CFNodeLeaf<dim>& leaf_iterator< dim >::operator* ( ) [inline]

Definition at line 26 of file leaf_iterator.h.

```
26 { return *leaf; }
```

5.13.3.3 template<boost::uint32_t dim> leaf_iterator leaf_iterator< dim >::operator++ ( ) [inline]

Definition at line 24 of file leaf_iterator.h.

```
24 { leaf = (CFNodeLeaf<dim>*)leaf->next.get(); return
     leaf_iterator(leaf); }
```

5.13.3.4 template<boost::uint32_t dim> CFNodeLeaf<dim>* leaf_iterator< dim >::operator-> ( ) [inline]

Definition at line 27 of file leaf_iterator.h.

```
27 { return leaf; }
```

5.13.4 Member Data Documentation

5.13.4.1 template<boost::uint32_t dim> CFNodeLeaf<dim>* leaf_iterator< dim >::leaf

Definition at line 29 of file leaf_iterator.h.

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

- leaf_iterator.h

5.14 subcluster_lessthan_norm Struct Reference

`#include <CFTree_Redist.h>`

Public Member Functions

- bool operator() (const subcluster_summary &lhs, const subcluster_summary &rhs) const
- bool operator() (const subcluster_summary &lhs, const subcluster_summary &rhs) const

5.14.1 Detailed Description

Definition at line 169 of file CFTree_Redist.h.

5.14.2 Member Function Documentation

5.14.2.1 bool subcluster_lessthan_norm::operator() (const subcluster_summary & *lhs*, const subcluster_summary & *rhs*) const [inline]

Definition at line 33 of file subcluster_summary.h.

```
33 { return (lhs.norm) < (rhs.norm); }
```

5.14.2.2 bool subcluster_lessthan_norm::operator() (const subcluster_summary & *lhs*, const subcluster_summary & *rhs*) const [inline]

Definition at line 171 of file CFTree_Redist.h.

```
171 { return (lhs.norm) < (rhs.norm); }
```

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

- CFTree_Redist.h
- subcluster_summary.h

5.15 subcluster_summary Struct Reference

`#include <CFTree_Redist.h>`

Public Member Functions

- subcluster_summary ()
- subcluster_summary (const ublas_vec_type &in_center, const float_type &in_radius, const float_type &in_norm)
- subcluster_summary ()
- subcluster_summary (const ublas_vec_type &in_center, const float_type &in_radius, const float_type &in_norm)

Public Attributes

- ublas_vec_type center
- float_type radius
- float_type norm

5.15.1 Detailed Description

Definition at line 156 of file CFTree_Redist.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 subcluster_summary::subcluster_summary ( ) [inline]

Definition at line 158 of file CFTree_Redist.h.

```
158 : radius(0.0), norm(0.0) {}
```

5.15.2.2 subcluster_summary::subcluster_summary (const ublas_vec_type & *in_center*, const float_type & *in_radius*, const float_type & *in_norm*) [inline]

Definition at line 159 of file CFTree_Redist.h.

```
159 : center( in_center ), radius(in_radius), norm(in_norm) {}
```

5.15.2.3 subcluster_summary::subcluster_summary ( ) [inline]

Definition at line 17 of file subcluster_summary.h.

```
17 : radius(0.0), norm(0.0) {}
```

5.15.2.4 subcluster_summary::subcluster_summary (const ublas_vec_type & *in_center,* const float_type & *in_radius,* const float_type & *in_norm*) [inline]

Definition at line 18 of file subcluster_summary.h.

```
20                                                  :
21                          center( in_center ),
22                          radius(in_radius),
23                          norm(in_norm) {}
```

5.15.3 Member Data Documentation

5.15.3.1 ublas_vec_type subcluster_summary::center

Definition at line 162 of file CFTree_Redist.h.

5.15.3.2 float_type subcluster_summary::norm

Definition at line 164 of file CFTree_Redist.h.

5.15.3.3 float_type subcluster_summary::radius

Definition at line 163 of file CFTree_Redist.h.

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

- CFTree_Redist.h
- subcluster_summary.h

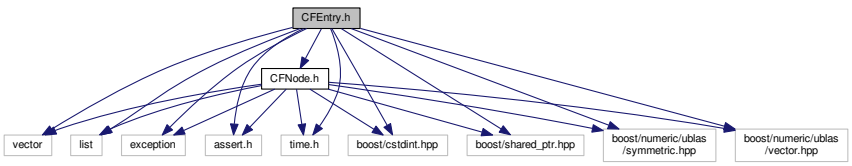
Chapter 6

File Documentation

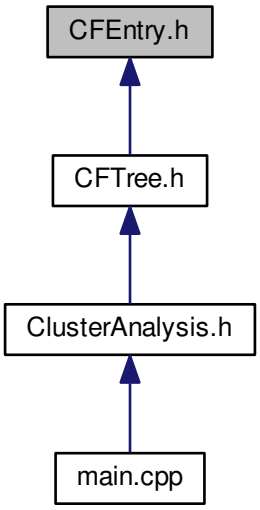
6.1 CFEntry.h File Reference

```
#include <vector>
#include <list>
#include <exception>
#include <assert.h>
#include <time.h>
#include <boost/cstdint.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/numeric/ublas/symmetric.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include "CFNode.h"
```

Include dependency graph for CFEntry.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct CFTreeInvalidItemSize
- class CFEntry< dim >

Typedefs

- typedef float float_type
- typedef std::vector< float_type > item_vec_type
- typedef boost::numeric::ublas::vector< float_type > ublas_vec_type
- typedef boost::numeric::ublas::symmetric_matrix< float_type > ublas_sym_matrix_type

6.1.1 Typedef Documentation

6.1.1.1 typedef float float_type

Definition at line 31 of file CFEntry.h.

6.1.1.2 typedef std::vector<float_type> item_vec_type

float type according to a precision - double, float, and so on.

Definition at line 32 of file CFEntry.h.

6.1.1.3 typedef boost::numeric::ublas::symmetric_matrix<float_type> ublas_sym_matrix_type

Definition at line 44 of file CFEntry.h.

6.1.1.4 typedef boost::numeric::ublas::vector<float_type> ublas_vec_type

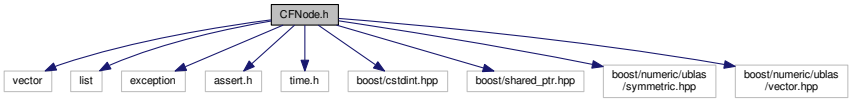
vector of items. pointer type of CFNode. shared_ptr is applied to preventing memory leakage. This node pointer is deleted, having no referencers

Definition at line 43 of file CFEntry.h.

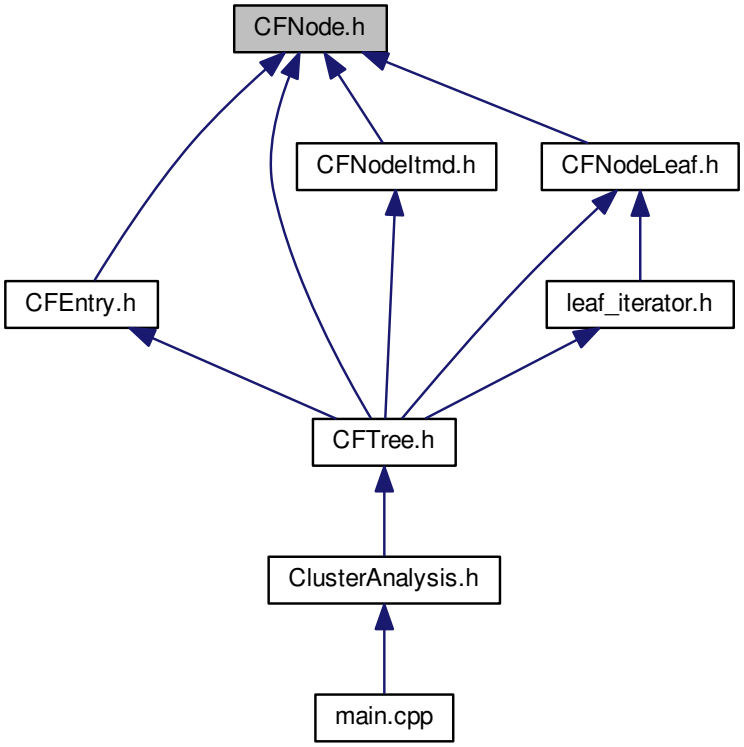
6.2 CFNode.h File Reference

```
#include <vector>
#include <list>
#include <exception>
#include <assert.h>
#include <time.h>
#include <boost/cstdint.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/numeric/ublas/symmetric.hpp>
#include <boost/numeric/ublas/vector.hpp>
```

Include dependency graph for CFNode.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CFEntry< dim >
- struct CFNodeLeaf< dim >
- struct CFNode< dim >

Macros

- #define PAGE_SIZE (500*1024) /* assuming 4K page */
- #define ARRAY_COUNT(a) (sizeof(a)/sizeof(a[0]))

6.2.1 Macro Definition Documentation

6.2.1.1 #define ARRAY_COUNT(*a*) (sizeof(a)/sizeof(a[0]))

Definition at line 27 of file CFNode.h.

6.2.1.2 #define PAGE_SIZE (500∗1024) /∗ assuming 4K page ∗/

CFNode is composed of several CFEntries within page-size, and acts like B-tree node.

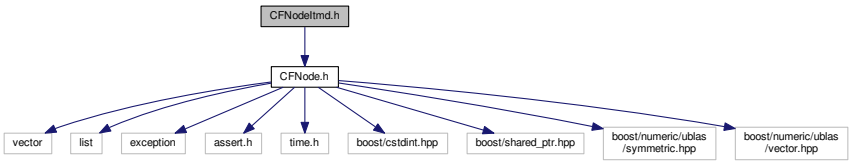
CFNode should be page-sized for more efficient operation. Like b-tree twist their node when removing and inserting node, CFTree perform similar operations on its own CFNodes.

CFNode has two types: intermediate node leaf node, especially leaf node has additional pointers to neighbor leaves.

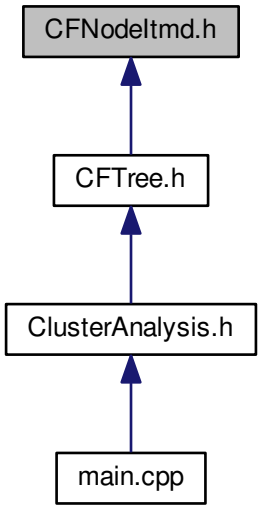
Definition at line 26 of file CFNode.h.

6.3 CFNodeItmd.h File Reference

`#include "CFNode.h"`
Include dependency graph for CFNodeItmd.h:



This graph shows which files directly or indirectly include this file:



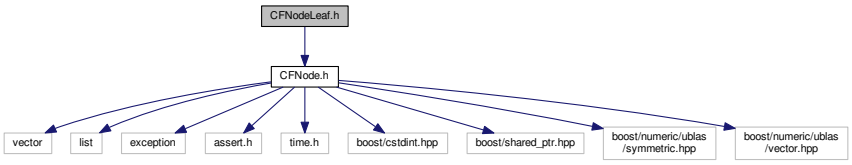
Classes

- struct CFNodeItmd< dim >

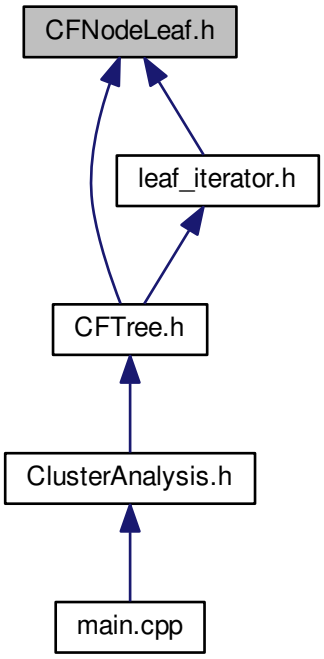
6.4 CFNodeLeaf.h File Reference

```
#include "CFNode.h"
```

Include dependency graph for CFNodeLeaf.h:



This graph shows which files directly or indirectly include this file:



Classes

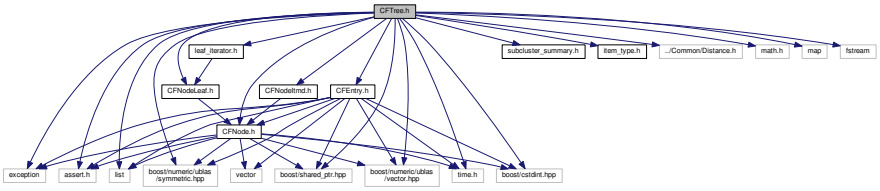
- struct CFNodeLeaf< dim >

6.5 CFTree.h File Reference

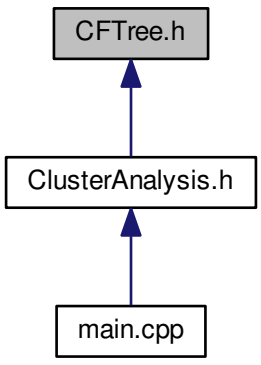
```
#include "CFEntry.h"
```

```
#include "CFNode.h"
#include "CFNodeItmd.h"
#include "CFNodeLeaf.h"
#include "leaf_iterator.h"
#include "subcluster_summary.h"
#include "item_type.h"
#include "../Common/Distance.h"
#include <math.h>
#include <map>
#include <list>
#include <fstream>
#include <exception>
#include <assert.h>
#include <time.h>
#include <boost/cstdint.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/numeric/ublas/symmetric.hpp>
#include <boost/numeric/ublas/vector.hpp>
```

Include dependency graph for CFTree.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CFTree< dim >
- struct CFTree< dim >::CFTreeInvalidItemSize
- struct CFTree< dim >::CloseEntryLessThan
- struct CFTree< dim >::HierarchicalClustering

6.6 CFTree_CFCluster.h File Reference

Classes

- struct HierarchicalClustering

Functions

- void cluster (cfentry_vec_type &entries)
- void refine_cluster (cfentry_vec_type &entries)
- void _cluster (cfentry_vec_type &entries)

6.6.1 Function Documentation

6.6.1.1 void _cluster (cfentry_vec_type & *entries*) [private]

Definition at line 324 of file CFTree_CFCluster.h.

```
325         {
326             int n = (int)entries.size();
327 
328             if( n <= 1 )
329                 return;
330 
331             if( dist_func == _DistD0 || dist_func == _DistD1 )
332             {
333                 refine_cluster( entries );
334             }
335             else
336             {
337                 HierarchicalClustering h( n - 1, dist_func );
338                 h.merge( entries );
339                 h.split( dist_threshold );
340                 h.result( entries );
341             }
342         }
```

6.6.1.2 void cluster (cfentry_vec_type & *entries*)

Definition at line 32 of file CFTree_CFCluster.h.

```
33         {
34             get_entries(entries);
35             _cluster(entries);
36         }
```

6.6.1.3 void refine_cluster (cfentry_vec_type & *entries*) [private]

Definition at line 259 of file CFTree_CFCluster.h.

```
260         {
261             std::vector<bool> merged(entries.size(), false);
262 
263             std::vector<std::size_t> not_visited;
264             not_visited.reserve( entries.size() );
265 
266             for( std::size_t i = 0 ; i < entries.size() ; i++ )
267                 not_visited.push_back(i);
268 
269             while( !not_visited.empty() )
270             {
271                 // pick any entry
272                 CFEntry& ref_entry = entries[not_visited.back()];
273                 CFEntry curr_entry = ref_entry;
274                 not_visited.pop_back();
275 
276                 bool something_merged = false;
277                 for( std::size_t i = 0 ; i < not_visited.size() ; i++ )
278                 {
279                     // index of next item
280                     std::size_t v = not_visited[i];
281                     CFEntry& e = entries[ v ];
282                     if( dist_func(ref_entry, e) <= dist_threshold/2 )
283                     {
284                         curr_entry += e;
285                         merged[ v ] = true;
286                         something_merged = true;
287                     }
288                 }
289 
290                 if( something_merged )
291                     ref_entry = curr_entry;
292 
293                 // remove if visited
294                 struct _remove_if_merged
295                 {
296                     _remove_if_merged( std::vector<bool>& in_visited ) : visited(in_visited) {}
297                     bool operator()( const std::size_t i )  const { return visited[i]; }
298                 private:
299                     std::vector<bool>& visited;
300                 };
301 
302                 // prepare for next loop, removing visited indices
303                 if( something_merged )
304                     not_visited.erase( std::remove_if( not_visited.begin(), not_visited.end(),
    _remove_if_merged(merged)), not_visited.end() );
305             }
306 
307             struct _remove_if_merged_by_item
308             {
309                 typedef CFEntry argument_type;
310                 _remove_if_merged_by_item( cfentry_vec_type& in_entries, std::vector<bool>& in_merged ) :
    entries(in_entries), merged(in_merged) {}
311                 bool operator() ( const CFEntry& e ) const
312                 {
313                     std::ptrdiff_t i = (&e - &entries[0]);
314                     return merged[i];
315                 }
316 
317             private:
318                 std::vector<bool>& merged;
319                 cfentry_vec_type& entries;
320             };
321             entries.erase( std::remove_if( entries.begin(), entries.end(), _remove_if_merged_by_item(
    entries ,merged) ), entries.end());
322         }
```

6.7 CFTree_Redist.h File Reference

Classes

- struct subcluster_summary
- struct subcluster_lessthan_norm

Typedefs

- typedef std::vector< subcluster_summary > subsum_vec_type

Functions

- bool _has_differences (std::vector< ublas_vec_type > &lhs, std::vector< ublas_vec_type > &rhs)
- bool _has_differences (std::vector< ublas_vec_type > &lhs, std::vector< ublas_vec_type > &rhs, std::vector< bool > &active)
- template<typename item_list_type >
  void redist_kmeans (item_list_type &items, cfentry_vec_type &entries, std::size_t iteration=2)
- template<typename _iter >
  void redist (_iter begin, _iter end, cfentry_vec_type &entries, std::vector< int > &out_cid)
- int _redist (ublas_vec_type &tmpv, subsum_vec_type &subsums, ublas_sym_matrix_type &dist_mat)

6.7.1 Typedef Documentation

6.7.1.1 typedef std::vector< subcluster_summary > subsum_vec_type

Definition at line 167 of file CFTree_Redist.h.

6.7.2 Function Documentation

6.7.2.1 bool _has_differences (std::vector< ublas_vec_type > & *lhs*, std::vector< ublas_vec_type > & *rhs*) [private]

Definition at line 27 of file CFTree_Redist.h.

```
28         {
29             assert( lhs.size() == rhs.size() );
30 
31             for( std::size_t i = 0 ; i < lhs.size() ; i++ )
32             {
33                 if( norm_2(lhs[i] - rhs[i]) > std::numeric_limits<float_type>::epsilon() )
34                     return true;
35             }
36             return false;
37         }
```

6.7.2.2 bool _has_differences (std::vector< ublas_vec_type > & *lhs*, std::vector< ublas_vec_type > & *rhs*, std::vector< bool > & *active*) [private]

Definition at line 39 of file CFTree_Redist.h.

```
40         {
41             assert( lhs.size() == rhs.size() );
42 
43             for( std::size_t i = 0 ; i < lhs.size() ; i++ )
44                 active[i] = norm_2(lhs[i] - rhs[i]) > std::numeric_limits<float_type>::epsilon();
45 
46             return std::count( active.begin(), active.end(), true ) > 0;
47         }
```

6.7.2.3 int _redist (ublas_vec_type & *tmpv,* subsum_vec_type & *subsums,* ublas_sym_matrix_type & *dist_mat*) [private]

Definition at line 219 of file CFTree_Redist.h.

```
220         {
221             int    imin,imax,i,k,n,start,end,median;
222             float d,tmpnorm,idist,tmpdist;
223             ublas_vec_type diff;
224 
225             i = 0;
226             n = (int)subsums.size() ;
227             tmpnorm = std::sqrt( inner_prod(tmpv, tmpv) );
228 
229             // i=ClosestNorm(tmpnorm,norms,0,n-1);
230             // for efficiency, replace recursion by iteration
231             start=0;
232             end=n-1;
233             while(start<end)
234             {
235                 if (end-start==1)
236                 {
237                     float_type norm_end = subsums[end].norm;
238                     float_type norm_start = subsums[start].norm;
239 
240                     i = tmpnorm > norm_end ? end :
241                         tmpnorm < norm_start ? start :
242                         tmpnorm - norm_start < norm_end - tmpnorm ? start : end;
243                     start = end = i;
244                 }
245                 else
246                 {
247                     median = (start+end)/2;
248                     float_type norm_med = subsums[median].norm;
249                     if (tmpnorm > norm_med)
250                         start=median;
251                     else
252                         end=median;
253                 }
254             }
255 
256             diff = tmpv - subsums[i].center;
257             idist= inner_prod(diff, diff);
258 
259             // imin=MinLargerThan(tmpnorm-sqrt(idist),norms,0,n-1);
260             // imax=MaxSmallerThan(tmpnorm+sqrt(idist),norms,0,n-1);
261             // for efficiency, replace recursion by iteration
262 
263             tmpdist=tmpnorm-sqrt(idist);
264             start=0;
265             end=n-1;
266             while (start<end)
267             {
268                 median=(start+end)/2;
269                 float_type norm_med = subsums[median].norm;
270                 if (tmpdist > norm_med)
271                     start=median+1;
272                 else
273                     end=median;
274             }
275             imin=start;
276 
277             tmpdist=tmpnorm+sqrt(idist);
278             start=0;
279             end=n-1;
280             while(start<end)
281             {
282                 median=(start+end+1)/2;
283                 float_type norm_med = subsums[median].norm;
284                 if (tmpdist < norm_med)
285                     end=median-1;
286                 else
287                     start=median;
288             }
289             imax=start;
290 
291             // ClosestCenter(i,idist,tmpv,centers,imin,imax,matrix,n);
292             // for efficiency, replace procedure by inline
293             k=imin;
294             while (k<=imax)
295             {
296                 if (dist_mat(k,i) <= 4*idist)
297                 {
```

```
298                     diff = tmpv - subsums[k].center;
299                     d = inner_prod(diff,diff);
300                     if (d < idist)
301                     {
302                         idist=d;
303                         i=k;
304                     }
305                 }
306                 k++;
307             }
308             return i;
309         }
```

6.7.2.4 template<typename _iter > void redist (_iter *begin*, _iter *end*, cfentry_vec_type & *entries*, std::vector< int > & *out_cid*)

Definition at line 175 of file CFTree_Redist.h.

```
176         {
177             using namespace boost::numeric::ublas;
178 
179             // prepare summaries for each subcluster
180             // summaries = ( center, radius, norm )
181             subsum_vec_type subclusters;
182             subclusters.reserve(entries.size());
183             for( std::size_t i = 0 ; i < entries.size() ; i++ )
184             {
185                 const CFEntry& e = entries[i];
186                 ublas_vec_type center(dim);
187                 std::copy(e.sum, e.sum + dim, center.begin());
188                 center /= e.n;
189                 subclusters.push_back( subcluster_summary( center, _Radius(e), std::sqrt(
    inner_prod(center, center) )) );
190             }
191 
192             std::sort( subclusters.begin(), subclusters.end(),
    subcluster_lessthan_norm() );
193 
194             // in addition to an individual summary for each subcluster
195             // calculate pairwise euclidean distances of subclusters
196             std::size_t n = subclusters.size();
197             ublas_sym_matrix_type dist_mat(n,n);
198             for( std::size_t i = 0 ; i < n-1 ; i++ )
199             {
200                 for( std::size_t j = i+1 ; j < n ; j++ )
201                 {
202                     ublas_vec_type diff = subclusters[i].center - subclusters[j].center;
203                     dist_mat(i,j) = inner_prod(diff, diff);
204                 }
205             }
206 
207             out_cid.clear();
208             out_cid.reserve(end - begin);
209             for( _iter it = begin ; it != end ; it++ )
210             {
211                 ublas_vec_type v(dim);
212                 std::copy(&(*it)[0], &(*it)[0] + dim, v.begin());
213                 out_cid.push_back( _redist( v, subclusters, dist_mat ) );
214             }
215         }
```

6.7.2.5 template<typename item_list_type > void redist_kmeans (item_list_type & *items*, cfentry_vec_type & *entries*, std::size_t *iteration* = 2)

Definition at line 52 of file CFTree_Redist.h.

```
53         {
54             using namespace boost::numeric::ublas;
55 
56             if( items.empty() )
57                 return;
```

```
58
59              assert(items[0].size() == dim);
60
61              // start from k means from k entries
62              std::vector<ublas_vec_type> prev_means(entries.size());
63              std::vector<ublas_vec_type> means(entries.size());
64              for( std::size_t i = 0 ; i < means.size() ; i++ )
65              {
66                  prev_means[i].resize( dim );
67                  prev_means[i].clear();
68
69                  CFEntry& e = entries[i];
70                  ublas_vec_type& mean = means[i];
71
72                  mean.resize( dim );
73                  std::copy( e.sum, e.sum + dim, mean.begin() );
74                  mean /= e.n;
75              }
76
77              // until it is converged
78              std::size_t iteration_count = 0;
79              if( iteration == 0 )
80                  iteration = (std::numeric_limits<std::size_t>::max)();
81
82              bool active = true;
83
84              //while( iteration_count < iteration && _has_differences( prev_means, means ) )
85              while( iteration_count < iteration && active )
86              {
87                  active = false;
88                  for( item_list_type::iterator item_it = items.begin() ; item_it != items.end() ; ++item_it
    )
89                  {
90                      item_list_type::value_type& item = *item_it;
91                      float_type min_dist = (std::numeric_limits<float_type>::max)();
92
93                      int prev_cid = item.cid();
94
95                      for( std::size_t cid = 0 ; cid < means.size() ; ++cid )
96                      {
97                          ublas_vec_type diff(dim);
98                          std::transform( &item[0], &item[0] + dim, means[cid].begin(), diff.begin(),
    std::minus<float_type>() );
99                          float_type dist = norm_2( diff );
100
101                          if( min_dist > dist )
102                          {
103                              min_dist = dist;
104                              item_it->cid() = cid;
105                          }
106                      }
107
108                      if( prev_cid != item.cid() )
109                          active = true;
110                  }
111
112                  std::stringstream ss;
113                  ss << "k-means_iteration" << iteration_count << ".txt";
114                  std::ofstream fout( ss.str().c_str() );
115
116                  for( std::size_t c = 0 ; c < means.size() ; c++ )
117                  {
118                      fout << "(" << c << ") ";
119                      for( std::size_t d = 0 ; d < dim ; d++ )
120                          fout << means[c][d] << (d == dim-1 ? "" : "," );
121                      fout << std::endl;
122                  }
123                  fout << std::endl;
124
125                  for( std::size_t i = 0 ; i < items.size() ; i++ )
126                      fout << i << ":" << items[i].cid() << std::endl;
127                  fout.close();
128
129                  // store means to prev_means and zeoring means
130                  prev_means = means;
131                  for( std::size_t i = 0 ; i < means.size() ; i++ )
132                      means[i].clear();
133
134                  // rearrange means and count # items for each cluster
135                  std::vector<std::size_t> mean_counts(means.size(), 0);
136                  for( item_list_type::iterator item_it = items.begin() ; item_it != items.end() ; ++item_it
    )
137                  {
138                      item_list_type::value_type& item = *item_it;
139                      std::transform( &item[0], &item[0] + dim, means[item.cid()].begin(), means[item.cid()].
    begin(), std::plus<float_type>() );
140                      ++mean_counts[item.cid()];
```

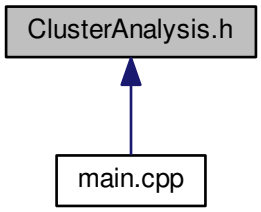
```
141                 }
142
143                 // averaging means to calculate centroids
144                 for( std::size_t i = 0 ; i < means.size() ; i++ )
145                     means[i] /= mean_counts[i];
146
147                 iteration_count++;
148             }
149             //std::cout << "iteration count = " << iteration_count << std::endl;
150         }
```

6.8 ClusterAnalysis.h File Reference

```
#include "CFTree.h"
#include "IOHandler.h"
#include "Silhouette.h"
#include "ValidityMeasurement.h"
#include <string>
#include <sstream>
#include <string.h>
#include <time.h>
```

Include dependency graph for ClusterAnalysis.h:

This graph shows which files directly or indirectly include this file:



Classes

- struct FileIndex

Typedefs

- typedef CFTree< 4824u > cftree_type

Functions

- template<boost::uint32_t dim>
  void getUserInput (const int &argc, char **argv, std::vector< std::vector< float > > &trajectories, Eigen::↩
  MatrixXf &equalArray, std::vector< item_type< dim > > &items, int &dimension, FileIndex &fi)
- template<typename T >
  static void print_items (const std::string fname, T &items)
- template<boost::uint32_t dim>
  static void load_items (const Eigen::MatrixXf &matrixData, std::vector< item_type< dim > > &items)
- const float getMaxDist (const Eigen::MatrixXf &equalArray, const MetricPreparation &object, const int &normOption)
- template<boost::uint32_t dim>
  void getBirchClusterTrial (const MetricPreparation &object, const int &normOption, std::vector< item_type< dim > > &items, const float &distThreshold, int &maxGroup, std::vector< int > &item_cids)
- template<boost::uint32_t dim>
  void getBirchClustering (std::vector< item_type< dim > > &items, char **argv, std::vector< std::vector< float > > &trajectories, const FileIndex &fi, Eigen::MatrixXf &equalArray, const int &dimension, std::vector< int > &item_cids, int &maxGroup, int &normOption, string &fullName, MetricPreparation &object)
- void getClusterAnalysis (const vector< vector< float > > &trajectories, const FileIndex &fi, const MatrixXf &equalArray, const int &dimension, vector< int > &item_cids, const int &maxGroup, const int &normOption, const string &fullName, const MetricPreparation &object)
- static float randf ()

Variables

- std::vector< string > activityList
- std::vector< double > timeList
- bool isPBF
- bool readCluster
- bool isPathlines
- cftree_type::float_type birch_threshold

6.8.1 Typedef Documentation

6.8.1.1 typedef CFTree<4824u> cftree_type

Definition at line 67 of file ClusterAnalysis.h.

6.8.2 Function Documentation

6.8.2.1 template<boost::uint32_t dim> void getBirchClustering (std::vector< item_type< dim > > & *items,* char ** *argv,* std::vector< std::vector< float > > & *trajectories,* const FileIndex & *fi,* Eigen::MatrixXf & *equalArray,* const int & *dimension,* std::vector< int > & *item_cids,* int & *maxGroup,* int & *normOption,* string & *fullName,* MetricPreparation & *object*)

Definition at line 345 of file ClusterAnalysis.h.

```
356 {
357     // select norm
358     std::cout << std::endl;
359     if(isPathlines)
360     {
361         std::cout << "Choose a norm from 0-17!" << std::endl;
362         std::cin >> normOption;
363         assert(normOption>=0 && normOption<=17);
364     }
365     else
366     {
367         std::cout << "Choose a norm from 0-16!" << std::endl;
368         std::cin >> normOption;
369         assert(normOption>=0 && normOption<=16);
370     }
371 
372     /*  0: Euclidean Norm
373         1: Fraction Distance Metric
374         2: piece-wise angle average
375         3: Bhattacharyya metric for rotation
376         4: average rotation
377         5: signed-angle intersection
378         6: normal-direction multivariate distribution
379         7: Bhattacharyya metric with angle to a fixed direction
380         8: Piece-wise angle average \times standard deviation
381         9: normal-direction multivariate un-normalized distribution
382         10: x*y/|x||y| borrowed from machine learning
383         11: cosine similarity
384         12: Mean-of-closest point distance (MCP)
385         13: Hausdorff distance min_max(x_i,y_i)
386         14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
387         15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
388         16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
389         17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
    S0097849318300128
390             for pathlines only
391     */
392 
393     struct timeval start, end;
394     double timeTemp;
395     gettimeofday(&start, NULL);
396 
397     // create MetricPreparation object
398     object = MetricPreparation(equalArray.rows(), equalArray.cols());
399     object.preprocessing(equalArray, equalArray.rows(), equalArray.cols(), normOption);
400 
401     // load the array into B+ tree
402     load_items(equalArray, items);
403     std::cout << items.size() << " items loaded" << std::endl;
404 
405     /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
406     if(!isPBF)
407     {
408         deleteDistanceMatrix(equalArray.rows());
409 
410         std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
411         if(distFile.fail()) // file does not exist, calculate distance matrix and store it into file
412         {
413             distFile.close();
414             getDistanceMatrix(equalArray, normOption, object);
415             std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
416             for(int i=0;i<equalArray.rows();++i)
417             {
418                 for(int j=0;j<equalArray.rows();++j)
419                 {
420                     distFileOut << distanceMatrix[i][j] << " ";
421                 }
422                 distFileOut << std::endl;
423             }
424             distFileOut.close();
425         }
426         else    // file exists, directly read in from the file
427         {
428             std::cout << "read distance matrix..." << std::endl;
429 
430             distanceMatrix = new float*[equalArray.rows()];
431         #pragma omp parallel for schedule(static) num_threads(8)
432             for (int i = 0; i < equalArray.rows(); ++i)
433             {
434                 distanceMatrix[i] = new float[equalArray.rows()];
435             }
436 
437             int i=0, j;
438             string line;
439             stringstream ss;
440             while(getline(distFile, line))
441             {
```

```
442                 j=0;
443                 ss.str(line);
444                 while(ss>>line)
445                 {
446                     if(i==j)
447                         distanceMatrix[i][j]=0;
448                     else
449                         distanceMatrix[i][j] = std::atof(line.c_str());
450                     ++j;
451                 }
452                 ++i;
453                 ss.str("");
454                 ss.clear();
455             }
456             distFile.close();
457         }
458     }
459     // get max distance
460     const float distThreshold = getMaxDist(equalArray, object, normOption);
461 
462     // get the input cluster number
463     int requiredClusters;
464     if(readCluster)  // from the file "cluster_number"
465     {
466         std::unordered_map<int,int> clusterMap;
467         IOHandler::readClusteringNumber(clusterMap, "cluster_number");
468         requiredClusters = clusterMap[normOption];
469     }
470     else    // or from user input on the console
471     {
472         std::cout << "Enter approximate number of clusters: " << std::endl;
473         std::cin >> requiredClusters;
474     }
475     const int& upperClusters = requiredClusters*1.2;
476     const int& lowerClusters = requiredClusters*0.8;
477 
478     std::cout << "Sampled max distance is: " << distThreshold << std::endl;
479 
480     float left = 0, right, middle;
481     if(normOption==15)
482         right = 0.2;
483     else
484         right = 0.5;
485 
486     // 10 times of binary search on the distance
487     int iteration = 0;
488     while(true&&iteration<10)
489     {
490         std::cout << "Iteration for birch clustering: " << ++iteration
491                   << std::endl;
492         middle = (left+right)/2.0;
493         std::cout << "Weight is " << middle << std::endl;
494         getBirchClusterTrial(object,normOption,items,middle*distThreshold,maxGroup,
    item_cids);
495         std::cout << maxGroup << std::endl;
496         if(maxGroup<=upperClusters && maxGroup>=lowerClusters)
497             break;
498         else if(maxGroup>upperClusters)
499             left = middle;
500         else if(maxGroup<lowerClusters)
501             right = middle;
502     }
503     birch_threshold = middle;
504 
505     // finish the birch clustering
506     gettimeofday(&end, NULL);
507     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
508                 + end.tv_usec - start.tv_usec) / 1.e6;
509     activityList.push_back("Birch clustering takes: ");
510     timeList.push_back(timeTemp);
511 
512     std::cout << "Max group is: " << maxGroup << std::endl;
513     //print_items(argc >= 4 ? ss.str().c_str() : "item_cid.txt", items);
514     stringstream ss;
515     ss << "../dataset/" << argv[1] << "_full.vtk";
516     fullName = ss.str();
517 
518     IOHandler::printVTK(fullName, trajectories, fi.vertexCount, dimension);
519 }
```

6.8.2.2 template<boost::uint32_t dim> void getBirchClusterTrial (const MetricPreparation & *object*, const int & *normOption*, std::vector< item_type< dim > > & *items*, const float & *distThreshold*, int & *maxGroup*, std::vector< int > & *item_cids*)

Definition at line 277 of file ClusterAnalysis.h.

```
283 {
284     cftree_type tree(distThreshold, 0);
285     tree.cftree_type::object = object;
286     tree.cftree_type::normOption = normOption;
287     tree.cftree_type::totalNodes = items.size();
288 
289     // phase 1 and 2: building, compacting when overflows memory limit
290     for( std::size_t i = 0 ; i < items.size() ; i++ )
291     {
292         if(&(items[i][0]))
293             tree.insert((float_type*)(&(items[i][0])));
294     }
295 
296     // phase 2 or 3: compacting? or clustering?
297     // merging overlayed sub-clusters by rebuilding true
298     std::cout << "Curve dimensionality is: " << cftree_type::fdim << std::endl;
299 
300     tree.rebuild(false);
301 
302     // phase 3: clustering sub-clusters using the existing clustering algorithm
303     //cftree_type::cfentry_vec_type entries;
304     std::vector<CFEntry<4824u> > entries;
305 
306     item_cids.clear();
307 
308     tree.cluster( entries );
309 
310     // phase 4: redistribution
311 
312     // @comment ts - it is also possible to another clustering algorithm hereafter
313     //              for example, we have k initial points for k-means clustering algorithm
314     //tree.redist_kmeans( items, entries, 0 );
315     tree.redist(items.begin(), items.end(), entries, item_cids);
316     maxGroup = INT_MIN;
317 
318     // assign the group labels
319     for (std::size_t i = 0; i < item_cids.size(); i++)
320     {
321         int& itemCID = items[i].cid();
322         itemCID = item_cids[i];
323         if(maxGroup<itemCID)
324             maxGroup=itemCID;
325     }
326 }
```

6.8.2.3 void getClusterAnalysis (const vector< vector< float > > & *trajectories*, const FileIndex & *fi*, const MatrixXf & *equalArray*, const int & *dimension*, vector< int > & *item_cids*, const int & *maxGroup*, const int & *normOption*, const string & *fullName*, const MetricPreparation & *object*)

Definition at line 535 of file ClusterAnalysis.h.

```
544 {
545     // get the size of clusters
546     int numClusters = maxGroup+1;
547     std::vector<int> container(numClusters,0);
548     for (int i = 0; i < item_cids.size(); ++i)
549         ++container[item_cids[i]];
550 
551     int increasingOrder[numClusters];
552     std::multimap<int,int> groupMap;
553 
554     for (int i = 0; i < numClusters; ++i)
555         groupMap.insert(std::pair<int,int>(container[i],i));
556 
557     // find how many clusters are formed
558     std::fill(container.begin(), container.end(), 0);
559     int groupNo = 0;
560     for (std::multimap<int,int>::iterator it=groupMap.begin();it!=groupMap.end();++it)
```

```
561     {
562         if(it->first>0)
563         {
564             increasingOrder[it->second] = groupNo;
565             container[groupNo] = it->first;
566             ++groupNo;
567         }
568     }
569 
570     numClusters = groupNo;
571     /* compute balanced Entropy value for the clustering algorithm */
572     const int& Row = equalArray.rows();
573     float entropy = 0.0, probability;
574     for(int i=0;i<container.size();++i)
575     {
576         probability = float(container[i])/float(Row);
577         entropy+=probability*log2f(probability);
578     }
579     entropy = -entropy/log2f(numClusters);
580 
581     // re-assign the cluster label in ascending order
582 #pragma omp parallel for schedule(static) num_threads(8)
583     for (int i = 0; i < item_cids.size(); ++i)
584         item_cids[i]=increasingOrder[item_cids[i]];
585 
586     std::vector<std::vector<int> > storage(numClusters);
587     for (int i = 0; i < item_cids.size(); ++i)
588         storage[item_cids[i]].push_back(i);
589 
590     /* record labeling information */
591     // IOHandler::generateGroups(storage);
592 
593 
594     IOHandler::printClusters(trajectories,item_cids,container,"norm"+to_string(normOption),
595             fullName,dimension);
596 
597     struct timeval start, end;
598     double timeTemp;
599 
600     // calculate the normalized validity measurement
601     ValidityMeasurement vm;
602     vm.computeValue(normOption, equalArray, item_cids, object, isPBF);
603     activityList.push_back("Validity measure is: ");
604     timeList.push_back(vm.f_c);
605 
606     // calculate the silhouette, gamma statistics and DB index
607     gettimeofday(&start, NULL);
608     Silhouette sil;
609     sil.computeValue(normOption,equalArray,equalArray.rows(),equalArray.cols(),item_cids,
610             object,numClusters,isPBF);
611     gettimeofday(&end, NULL);
612     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
613                 + end.tv_usec - start.tv_usec) / 1.e6;
614     activityList.push_back("Silhouette calculation takes: ");
615     timeList.push_back(timeTemp);
616 
617     /* compute the centroid coordinates of each clustered group */
618     Eigen::MatrixXf centroid = MatrixXf::Zero(numClusters,equalArray.cols());
619     vector<vector<float> > cenVec(numClusters);
620 #pragma omp parallel for schedule(static) num_threads(8)
621     for (int i=0;i<numClusters;++i)
622     {
623         const std::vector<int>& groupRow = storage[i];
624         for (int j = 0; j < groupRow.size(); ++j)
625         {
626             centroid.row(i)+=equalArray.row(groupRow[j]);
627         }
628         centroid.row(i)/=groupRow.size();
629         const Eigen::VectorXf& vec = centroid.row(i);
630         cenVec[i] = vector<float>(vec.data(), vec.data()+equalArray.cols());
631     }
632 
633     // calculate the closest and further representative for each cluster
634     vector<vector<float> > closest(numClusters);
635     vector<vector<float> > furthest(numClusters);
636 #pragma omp parallel for schedule(static) num_threads(8)
637     for (int i=0;i<numClusters;++i)
638     {
639         float minDist = FLT_MAX;
640         float maxDist = -10;
641         int minIndex = -1, maxIndex = -1;
642         const std::vector<int>& groupRow = storage[i];
643         const Eigen::VectorXf& eachCentroid = centroid.row(i);
644         for (int j = 0; j < groupRow.size(); ++j)
645         {
646             float distance = getDisimilarity(eachCentroid,equalArray,groupRow[j],normOption,object);
647             if(minDist>distance)
```

```
648             {
649                 minDist = distance;
650                 minIndex = groupRow[j];
651             }
652             if(maxDist<distance)
653             {
654                 maxDist = distance;
655                 maxIndex = groupRow[j];
656             }
657         }
658         closest[i] = trajectories[minIndex];
659         furthest[i] = trajectories[maxIndex];
660     }
661 
662     // print the representative in .vtk format
663     std::cout << "Finishing extracting features!" << std::endl;
664     IOHandler::printFeature("norm"+to_string(normOption)+"_closest.vtk",
665             closest, sil.sCluster, dimension);
666     IOHandler::printFeature("norm"+to_string(normOption)+"_furthest.vtk",
667             furthest, sil.sCluster, dimension);
668     IOHandler::printFeature("norm"+to_string(normOption)+"_centroid.vtk",
669             cenVec, sil.sCluster,dimension);
670 
671     IOHandler::printToFull(trajectories, sil.sData,
672             "norm"+to_string(normOption)+"_SValueLine", fullName, dimension);
673     IOHandler::printToFull(trajectories, item_cids, sil.sCluster,
674               "norm"+to_string(normOption)+"_SValueCluster", fullName, dimension);
675 
676     IOHandler::generateReadme(activityList,timeList,normOption,
677                 numClusters,sil.sAverage,birch_threshold);
678 
679     /* print entropy value for the clustering algorithm */
680     IOHandler::writeReadme(entropy,sil,"For norm "+to_string(normOption));
681 
682     /* measure closest and furthest rotation */
683     std::vector<float> closestRot, furthestRot;
684     const float& closestAverage = getRotation(closest, closestRot);
685     const float& furthestAverage = getRotation(furthest, furthestRot);
686 
687     IOHandler::writeReadme(closestAverage, furthestAverage);
688 }
```

6.8.2.4 const float getMaxDist (const Eigen::MatrixXf & *equalArray*, const MetricPreparation & *object*, const int & *normOption*)

Definition at line 237 of file ClusterAnalysis.h.

```
240 {
241     // find the maximal distance value
242     const float& Percentage = 0.1;
243     const int& chosen = int(Percentage*equalArray.rows());
244     float result = -0.1;
245 #pragma omp parallel for reduction(max:result) num_threads(8)
246     for (int i = 0; i < chosen; ++i)
247     {
248         for (int j = 0; j < equalArray.rows(); ++j)
249         {
250             if(i==j)
251                 continue;
252 
253             float dist;
254             if(distanceMatrix)
255                 dist = distanceMatrix[i][j];
256             else
257                 dist = getDisimilarity(equalArray.row(i), equalArray.row(j),i,j,normOption,object);
258             if(dist>result)
259                 result=dist;
260         }
261     }
262     return result;
263 }
```

6.8.2.5 template<boost::uint32_t dim> void getUserInput (const int & *argc*, char ** *argv*, std::vector< std::vector< float > > & *trajectories*, Eigen::MatrixXf & *equalArray*, std::vector< item_type< dim > > & *items*, int & *dimension*, FileIndex & *fi*)

Definition at line 100 of file ClusterAnalysis.h.

```
107 {
108     if( argc != 3 )
109     {
110         std::cout << "usage: birch (input-file) (dimension)" << std::endl;
111         exit(1);
112     }
113     int samplingMethod;
114     stringstream ss;
115     ss << "../dataset/" << argv[1];
116 
117     /* get the bool tag for isPBF */
118     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
119     int PBFjudgement;
120     std::cin >> PBFjudgement;
121     assert(PBFjudgement==1||PBFjudgement==0);
122     isPBF = (PBFjudgement==1);
123 
124     // whether it is pathline
125     std::cout << "It is a pathline dataset? 1.Yes, 0.No" << std::endl;
126     std::cin >> PBFjudgement;
127     assert(PBFjudgement==1||PBFjudgement==0);
128     isPathlines = (PBFjudgement==1);
129 
130     // how to get input number of clusters
131     std::cout << "---------------------------" << std::endl;
132     std::cout << "Choose cluster number input method: 0.user input, 1.read from file: " << std::endl;
133     int clusterInput;
134     std::cin >> clusterInput;
135     assert(clusterInput==0||clusterInput==1);
136     readCluster = (clusterInput==1);
137 
138     // sampling
139     if(isPathlines)
140         samplingMethod = 1;
141     else
142     {
143         std::cout << "Please choose the sampling method? " << endl
144                   << "1.filling, 2.uniform sampling." << std::endl;
145         std::cin >> samplingMethod;
146     }
147     assert(samplingMethod==1||samplingMethod==2);
148 
149     dimension = atoi(argv[2]);
150 
151     struct timeval start, end;
152     double timeTemp;
153 
154     // read coordinates from the txt file
155     gettimeofday(&start, NULL);
156     IOHandler::readFile(ss.str(), trajectories, fi.vertexCount, dimension, fi.
    maxElement);
157     gettimeofday(&end, NULL);
158     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
159                 + end.tv_usec - start.tv_usec) / 1.e6;
160     activityList.push_back("Read file takes: ");
161     timeList.push_back(timeTemp);
162 
163     // perform sampling on the data sets
164     gettimeofday(&start, NULL);
165     if(samplingMethod==1)
166         IOHandler::expandArray(equalArray,trajectories,dimension,
167                                fi.maxElement);
168     else if(samplingMethod==2)
169         IOHandler::sampleArray(equalArray,trajectories,dimension,
170                                fi.maxElement);
171     gettimeofday(&end, NULL);
172     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
173                 + end.tv_usec - start.tv_usec) / 1.e6;
174     activityList.push_back("Pre-processing takes: ");
175     timeList.push_back(timeTemp);
176 }
```

6.8.2.6 template<boost::uint32_t dim> static void load_items (const Eigen::MatrixXf & *matrixData,* std::vector< item_type< dim > > & *items*) [static]

Definition at line 215 of file ClusterAnalysis.h.

```
217 {
218     items.resize(matrixData.rows());
219 #pragma omp parallel for schedule(static) num_threads(8)
220     for (int i = 0; i < items.size(); ++i)
221     {
222         const Eigen::VectorXf& eachRow = matrixData.row(i);
223         cftree_type::item_vec_type item(eachRow.data(), eachRow.data()+eachRow.
    size());
224         items[i] = &(item[0]);
225     }
226 }
```

6.8.2.7 template<typename T > static void print_items (const std::string *fname,* T & *items*) [static]

Definition at line 186 of file ClusterAnalysis.h.

```
187 {
188     struct _compare_item_id
189     {
190         bool operator()( const item_type<3>& lhs, const item_type<3>& rhs )
191         const { return lhs.cid() < rhs.cid(); }
192     };
193
194     // find the max group index
195     int maxGroup = INT_MIN;
196     int belongGroup;
197     for( std::size_t i = 0 ; i < items.size() ; i++ )
198     {
199 //      for( std::size_t d = 0 ; d < cftree_type::fdim ; d++ )
200 //          fout << items[i].item[d] << " ";
201         belongGroup = items[i].cid();
202         if(belongGroup>maxGroup)
203             maxGroup=belongGroup;
204     }
205 }
```

6.8.2.8 static float randf ( ) [static]

Definition at line 695 of file ClusterAnalysis.h.

```
696 {
697     return float(rand()/(double)RAND_MAX);
698 }
```

6.8.3 Variable Documentation

6.8.3.1 std::vector<string> activityList

Definition at line 28 of file ClusterAnalysis.h.

6.8.3.2 cftree_type::float_type birch_threshold

Definition at line 72 of file ClusterAnalysis.h.

6.8.3.3 bool isPathlines

Definition at line 44 of file ClusterAnalysis.h.

6.8.3.4 bool isPBF

Definition at line 34 of file ClusterAnalysis.h.

6.8.3.5 bool readCluster

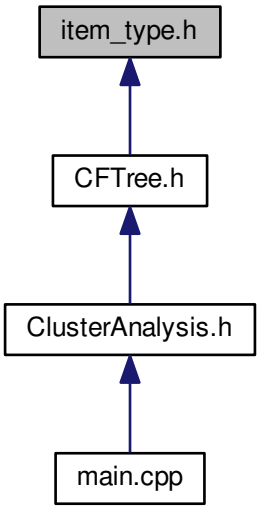
Definition at line 39 of file ClusterAnalysis.h.

6.8.3.6 std::vector<double> timeList

Definition at line 29 of file ClusterAnalysis.h.

6.9 item_type.h File Reference

This graph shows which files directly or indirectly include this file:



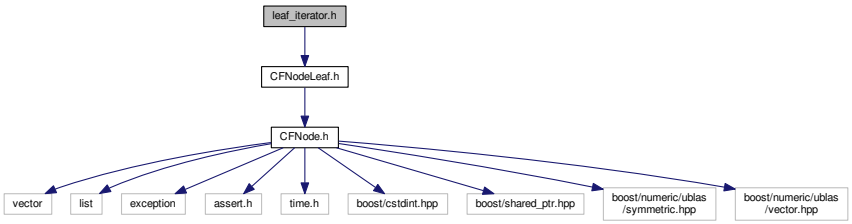
Classes

- class CFTree< dim >
- struct item_type< dim >

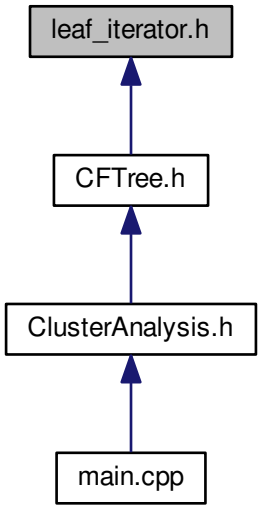
6.10 leaf_iterator.h File Reference

```
#include "CFNodeLeaf.h"
```

Include dependency graph for leaf_iterator.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct leaf_iterator< dim >

Typedefs

- typedef std::random_access_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

6.10.1 Typedef Documentation

6.10.1.1 typedef std::ptrdiff_t difference_type

Definition at line 18 of file leaf_iterator.h.

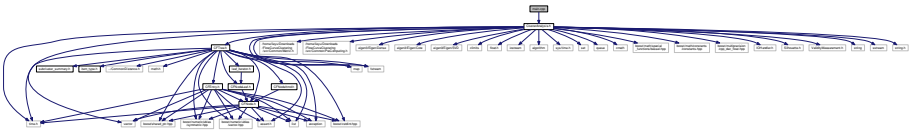
6.10.1.2 typedef std::random_access_iterator_tag iterator_category

Definition at line 12 of file leaf_iterator.h.

6.11 main.cpp File Reference

```
#include "ClusterAnalysis.h"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char **argv)

6.11.1 Function Documentation

6.11.1.1 int main (int *argc,* char ** *argv*)

Simple test code for birch-clustering algorithm

BIRCH has 4 phases: building, compacting, clustering, redistribution.

building - building cftree inserting a new data-point compacting - make cftree smaller enlarging the range of sub-clusters clustering - clustering sub-clusters(summarized clusters) using the existing clustering algorithm redistribution - labeling data-points to the closest center

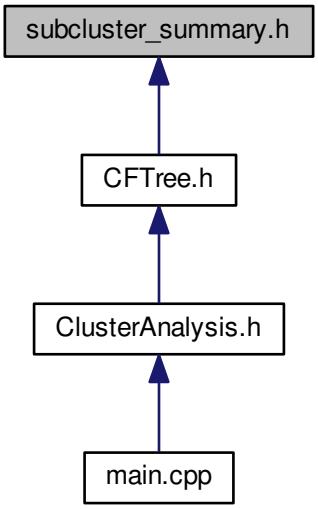
Definition at line 42 of file main.cpp.

```
43 {
44     std::vector<std::vector<float> > trajectories;
45     Eigen::MatrixXf equalArray;
46     std::vector<item_type<4824u> > items;
47     int dimension, maxGroup, normOption;
48     FileIndex fi;
49     std::vector<int> item_cids;
50     string fullName;
51     MetricPreparation object;
52 
53     // get user input for birch clustering
54     getUserInput(argc, argv, trajectories, equalArray, items, dimension, fi);
55 
56     // perform birch clustering
57     getBirchClustering(items,argv,trajectories,fi,equalArray, dimension, item_cids,
       maxGroup, normOption,
58              fullName, object);
59 
60     // perform the clustering analysis and cluster representative extraction
61     getClusterAnalysis(trajectories, fi, equalArray, dimension, item_cids, maxGroup,
       normOption,
62              fullName, object);
63     return 0;
64 }
```

6.12 README.md File Reference

6.13 subcluster_summary.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct subcluster_summary
- struct subcluster_lessthan_norm

Index