N2

Release 0.1.7

Kakao Recommendation Team

Oct 16, 2020

GETTING STARTED

Lightweight approximate **N**earest **N**eighbor algorithm library written in C++ (with Python/Go bindings).

N2 stands for two N's, which comes from 'Approximate Nearest Neighbor Algorithm'.

CHAPTER

ONE

WHY N2 WAS MADE

Before N2, there has been other great approximate nearest neighbor libraries such as Annoy and NMSLIB. However, each of them had different strengths and weaknesses regarding usability, performance, and etc. So, N2 has been developed aiming to bring the strengths of existing aKNN libraries and supplement their weaknesses.

CHAPTER

TWO

FEATURES

- Lightweight library which runs fast with large datasets.
- Good performance in terms of index build time, search speed, and memory usage.
- Supports multi-core CPUs for index building.
- Supports a mmap feature by default to efficiently process large index files.
- Supports Python/Go bindings.

CHAPTER

THREE

SUPPORTED DISTANCE METRICS

Metric	Definition	$d(\vec{p}, \vec{q})$
"angular"	$1 - \cos\theta$	$1 - \frac{\vec{p} \cdot \vec{q}}{\|\|\vec{p}\|\| \, \|\|\vec{q}\|\|} = 1 - \frac{\sum_i p_i q_i}{\sqrt{\sum_i p_i^2}\sqrt{\sum_i q_i^2}}$
"L2"	squared L2	$\sum_i (p_i - q_i)^2$
"dot"	dot product	$\vec{p} \cdot \vec{q} = \sum_i p_i q_i$

N2 supports three distance metrics. For "angular" and "L2", **d** (distance) is defined such that the closer the vectors are, the smaller **d** is. However for "dot", **d** is defined such that the closer the vectors are, the larger **d** is. You may be wondering why we defined and implemented "dot" metric as *plain dot product* and not as *(1 - dot product)*. The rationale for this decision was to allow users to directly interpret the **d** value returned from Hnsw search function as a dot product value.

3.1 Installation

`master` branch is always the latest release version of N2 and `dev` branch is the development branch for the next release.

3.1.1 Requirements

- gcc
- openmp

Note: Note that you must install gcc that supports C++14. For macOS users, please ensure that gcc is installed with brew. Currently, N2 build is not supported for gcc linked to Clang.

Note: Regardless of your language choice (Python, C++, Go), any command described in the section `Install from source` should be run from the root of N2 directory, assumming that you have successfully run the following commands.

```
$ git clone https://github.com/kakao/n2.git
$ cd n2
$ git submodule update --init  # update submodules
```

3.1.2 Python

You can install N2 using pip or directly from source.

Install using pip

The easiest way to install N2 is to use pip. This will automatically install Cython dependency.

```
$ pip install n2
```

Install from source

Or you can build from source by running the following command.

```
$ python setup.py install
```

You can run unit test with:

```
$ make test_python
```

3.1.3 C++

Install from source

1. Depending on what you want, run either of the following commands:

   ```
   $ make shared_lib  # If you need shared library
   ```

   ```
   $ make static_lib  # If you need static library
   ```

2. You can install N2 shared library (built with `make shared_lib`) into user-defined location set by PREFIX environment variable with the following command:

   ```
   $ make install  # Default installation path is /usr/local/.
   ```

3. You can run unit test with:

   ```
   $ make test_cpp
   ```

3.1.4 Go

Install from source

```
# Set GOPATH first!
$ make go
```

3.1.5 Installation FAQ

I'm having trouble installing N2 on macOS.

After you install gcc with brew, `python setup.py install` will work fine. But `make shared_lib` or `make static_lib` can still produce errors similar to the following:

```
$ make shared_lib
cd src/ && make shared_lib && cd .. && mkdir -p build/lib && \
      mv src/libn2.so ./build/lib/libn2.so && \
      cp build/lib/libn2.so build/lib/libn2.so.0.1.6
c++ -O3 -march=native -std=c++14 -pthread -fPIC -fopenmp -DNDEBUG -DBOOST_DISABLE_
↪ASSERTS
-I../third_party/spdlog/include/ -I../include/ -I../third_party/eigen -I../third_
↪party/boost/assert/include/
-I../third_party/boost/bind/include/ -I../third_party/boost/concept_check/include/
-I../third_party/boost/config/include/ -I../third_party/boost/core/include/ -I../
↪third_party/boost/detail/include/
-I../third_party/boost/heap/include/ -I../third_party/boost/iterator/include/ -I../
↪third_party/boost/mp11/include/
-I../third_party/boost/mpl/include/ -I../third_party/boost/parameter/include/
-I../third_party/boost/preprocessor/include/ -I../third_party/boost/static_assert/
↪include/
-I../third_party/boost/throw_exception/include/ -I../third_party/boost/type_traits/
↪include/
-I../third_party/boost/utility/include/   -c -o hnsw.o hnsw.cc
clang: error: unsupported option '-fopenmp'
make[1]: *** [hnsw.o] Error 1
make: *** [shared_lib] Error 2
```

In this case, possible reason is that you have not properly set symbolic links or environment variables to point to brew-installed gcc. Thus, please make sure that gcc/g++ symbolic links are linked to brew-installed gcc, or CC/CXX environment variables are set as brew-installed gcc/g++. There may be other solutions and here is one possible fix to this problem.

```
# Set CC, CXX environment variables
$ export CC=$(find $(brew --prefix gcc)/bin -type f -name 'gcc-[0-9]*')
$ export CXX=$(find $(brew --prefix gcc)/bin -type f -name 'g++-[0-9]*')
```

3.2 Python Interface

3.2.1 Basic Usage

```
import random

from n2 import HnswIndex

f = 40
t = HnswIndex(f)  # HnswIndex(f, "angular, L2, or dot")
for i in range(1000):
    v = [random.gauss(0, 1) for z in range(f)]
    t.add_data(v)

t.build(m=5, max_m0=10, n_threads=4)
```

(continues on next page)

(continued from previous page)

```
t.save('test.hnsw')

u = HnswIndex(f)
u.load('test.hnsw')
print(u.search_by_id(0, 1000))
```

You can see more code examples at examples/python.

3.2.2 Main Interface

n2.HnswIndex.add_data	Adds vector v.
n2.HnswIndex.build	Builds a hnsw graph with given configurations.
n2.HnswIndex.save	Saves the index to disk.
n2.HnswIndex.load	Loads the index from disk.
n2.HnswIndex.unload	Unloads (unmap) the index.
n2.HnswIndex.search_by_vector	Returns k nearest items (as vectors) to a query item.
n2.HnswIndex.search_by_id	Returns k nearest items (as ids) to a query item.
n2.HnswIndex.batch_search_by_vectors	Returns k nearest items (as vectors) to each query item (batch search with multi-threads).
n2.HnswIndex.batch_search_by_ids	Returns k nearest items (as ids) to each query item (batch search with multi-threads).

class n2.**HnswIndex**(*dimension*, *metric='angular'*)

__init__(*dimension*, *metric='angular'*)

Parameters

- **dimension** (*int*) -- Dimension of vectors.
- **metric** (*string*) -- An optional parameter to choose a distance metric. ('angular' | 'L2' | 'dot')

Returns An instance of Hnsw index.

add_data(*v*)

Adds vector v.

Parameters **v** (*list(float)*) -- A vector with dimension `dimension` set in __init__().

Returns Boolean value indicating whether data addition succeeded or not.

Return type bool

batch_search_by_ids(*item_ids*, *k*, *ef_search=- 1*, *num_threads=4*, *include_distances=False*)

Returns k nearest items (as ids) to each query item (batch search with multi-threads).

Note: With OMP_SCHEDULE environment variable, you can set how threads are scheduled. Refer to GNU libgomp.

Parameters

- **item_ids** (*list(int)*) -- Query ids.

- **k** (*int*) -- k value.
- **ef_search** (*int*) -- ef_search metric (default: 50 * k). If you pass -1 to ef_search, ef_search will be set as the default value.
- **num_threads** (*int*) -- Number of threads to use for search.
- **include_distances** (*bool*) -- If you set this argument to True, it will return a list of tuples((item_id, distance)).

Returns A list of list of k nearest items for each input query item in the order passed to parameter `item_ids`.

Return type list(list(int) or list(list(tuple(int, float)))

batch_search_by_vectors(*vs*, *k*, *ef_search=- 1*, *num_threads=4*, *include_distances=False*)
Returns k nearest items (as vectors) to each query item (batch search with multi-threads).

Note: With OMP_SCHEDULE environment variable, you can set how threads are scheduled. Refer to GNU libgomp.

Parameters

- **vs** (*list(list(float))*) -- Query vectors.
- **k** (*int*) -- k value.
- **ef_search** (*int*) -- ef_search metric (default: 50 * k). If you pass -1 to ef_search, ef_search will be set as the default value.
- **num_threads** (*int*) -- Number of threads to use for search.
- **include_distances** (*bool*) -- If you set this argument to True, it will return a list of tuples((item_id, distance)).

Returns A list of list of k nearest items for each input query item in the order passed to parameter `vs`.

Return type list(list(int) or list(list(tuple(int, float)))

build(*m=None*, *max_m0=None*, *ef_construction=None*, *n_threads=None*, *mult=None*, *neighbor_selecting=None*, *graph_merging=None*)
Builds a hnsw graph with given configurations.

Parameters

- **m** (*int*) -- Max number of edges for nodes at level > 0 (default: 12).
- **max_m0** (*int*) -- Max number of edges for nodes at level == 0 (default: 24).
- **ef_construction** (*int*) -- Refer to HNSW paper (default: 150).
- **n_threads** (*int*) -- Number of threads for building index.
- **mult** (*float*) -- Level multiplier. Recommended to use the default value (default: 1 / log(1.0 * M)).
- **neighbor_selecting** (*string*) -- Neighbor selecting policy.
  - **Available values**
    - `"heuristic"` (default): Select neighbors using algorithm4 on HNSW paper (recommended).

* `"naive"`: Select closest neighbors (not recommended).

- **graph_merging** (*string*) -- Graph merging heuristic.
  - **Available values**
    * `"skip"` (default): Do not merge (recommended for large-scale data (over 10M)).
    * `"merge_level0"`: Performs an additional graph build in reverse order, then merges edges at level 0. So, it takes twice the build time compared to `"skip"` but shows slightly higher accuracy. (recommended for data under 10M scale).

load (*fname*, *use_mmap=True*)
Loads the index from disk.

Parameters

- **fname** (*str*) -- An index file name.
- **use_mmap** (*bool*) -- An optional parameter indicating whether to use mmap() or not (default: True). If this parameter is set, N2 loads model through mmap.

Returns Boolean value indicating whether model load succeeded or not.

Return type bool

save (*fname*)
Saves the index to disk.

Parameters **fname** (*str*) -- A file destination where the index will be saved.

Returns Boolean value indicating whether model save succeeded or not.

Return type bool

search_by_id (*item_id*, *k*, *ef_search=- 1*, *include_distances=False*)
Returns k nearest items (as ids) to a query item.

Parameters

- **item_id** (*int*) -- A query id.
- **k** (*int*) -- k value.
- **ef_search** (*int*) -- ef_search metric (default: 50 * k). If you pass -1 to ef_search, ef_search will be set as the default value.
- **include_distances** (*bool*) -- If you set this argument to True, it will return a list of tuples((item_id, distance)).

Returns A list of k nearest items.

Return type list(int) or list(tuple(int, float))

search_by_vector (*v*, *k*, *ef_search=- 1*, *include_distances=False*)
Returns k nearest items (as vectors) to a query item.

Parameters

- **v** (*list(float)*) -- A query vector.
- **k** (*int*) -- k value.
- **ef_search** (*int*) -- ef_search metric (default: 50 * k). If you pass -1 to ef_search, ef_search will be set as the default value.

- **include_distances** (*bool*) -- If you set this argument to True, it will return a list of tuples((item_id, distance)).

Returns A list of k nearest items.

Return type list(int) or list(tuple(int, float))

unload()
Unloads (unmap) the index.

3.3 C++ Interface

3.3.1 Basic Usage

```cpp
#include "hnsw.h"

#include <utility>
#include <vector>

int main() {
    n2::Hnsw index(3, "angular");
    index.AddData(std::vector<float>{0, 0, 1});
    index.AddData(std::vector<float>{0, 1, 0});
    index.AddData(std::vector<float>{0, 0, 1});

    int n_threads = 4;
    int m = 5;
    int max_m0 = 10;
    int ef_construction = 50;
    index.Build(m, max_m0, ef_construction, n_threads);
    std::vector<std::pair<int, float>> result;
    int ef_search = 30;
    index.SearchByVector(std::vector<float>{3, 2, 1}, 3, ef_search, result);
    return 0;
}
```

You can see more code examples at examples/cpp.

3.3.2 Main Interface

class n2::**Hnsw**

Public Functions

Hnsw()

Hnsw(int *dim*, std::string *metric* = "angular")
Makes an instance of *Hnsw* Index.

Return A new *Hnsw* index.

Parameters

- dim: Dimension of vectors.

- `metric`: An optional parameter to choose a distance metric. ('angular' | 'L2' | 'dot') (default: 'angular').

Hnsw(**const** *Hnsw* &*other*)

Hnsw(*Hnsw* &&*other*) **noexcept**

void **AddData**(**const** std::vector<float> &*data*)
Adds vector to *Hnsw* index.

Parameters

- `data`: A vector with dimension `dim`.

void **SetConfigs**(**const** std::vector<std::pair<std::string, std::string>> &*configs*)
Set configurations by key/value pairs.

To set configurations as default values, pass negative values to configuration parameters.

void **Build**(int *m* = -1, int *max_m0* = -1, int *ef_construction* = -1, int *n_threads* = -1, float *mult* = -1, *NeighborSelectingPolicy* *neighbor_selecting* = *NeighborSelectingPolicy*::*HEURISTIC*, *GraphPostProcessing* *graph_merging* = *GraphPostProcessing*::*SKIP*, bool *ensure_k* = false)
Builds a hnsw graph with given configurations.

To see other available values for `neighbor_selecting` and `graph_merging`, refer to *NeighborSelectingPolicy()* and *GraphPostProcessing()*.

See *Fit()*, *SetConfigs()*

Parameters

- `m`: Max number of edges for nodes at level > 0 (default: 12).
- `max_m0`: Max number of edges for nodes at level == 0 (default: 24).
- `ef_construction`: Refer to HNSW paper for its role (default: 150).
- `n_threads`: Number of threads for building index.
- `mult`: Level multiplier (default value recommended) (default: 1/log(1.0*M)).
- `NeighborSelectingPolicy`: Neighbor selecting policy.
- `GraphPostProcessing`: Graph merging heuristic.

void **Fit**()
Builds a hnsw graph with given configurations.

bool **SaveModel**(**const** std::string &*fname*) **const**
Saves the index to disk.

Parameters

- `fname`: An index file name.

bool **LoadModel**(**const** std::string &*fname*, **const** bool *use_mmap* = true)
Loads an index from disk.

Parameters

- `fname`: An index file name.

- `use_mmap`: An optional parameter (default: true). If this parameter is set, N2 loads model through mmap.

void **`UnloadModel`**`()`
Unloads the loaded index file.

void **`SearchByVector`**`(`**`const`** std::vector<float> &*qvec*, size_t *k*, size_t *ef_search*, std::vector<int> &*result*`)`

void **`SearchByVector`**`(`**`const`** std::vector<float> &*qvec*, size_t *k*, size_t *ef_search*, std::vector<std::pair<int, float>> &*result*`)`
Search k nearest items (as vectors) to a query item.

Parameters

- `qvec`: A query vector.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `[out] result`: `k` nearest items.

void **`SearchById`**`(`int *id*, size_t *k*, size_t *ef_search*, std::vector<int> &*result*`)`

void **`SearchById`**`(`int *id*, size_t *k*, size_t *ef_search*, std::vector<std::pair<int, float>> &*result*`)`
Search k nearest items (as ids) to a query item.

Parameters

- `id`: A query id.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `[out] result`: `k` nearest items.

void **`BatchSearchByVectors`**`(`**`const`** std::vector<std::vector<float>> &*qvecs*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<int>> &*results*`)`

void **`BatchSearchByVectors`**`(`**`const`** std::vector<std::vector<float>> &*qvecs*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<std::pair<int, float>>> &*results*`)`
Search k nearest items (as vectors) to each query item (batch search with multi-threads).

Parameters

- `qvecs`: Query vectors.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `n_threads`: Number of threads to use for search.
- `[out] result`: vector of `k` nearest items for each input query item in the order passed to parameter `qvecs`.

void **`BatchSearchByIds`**`(`**`const`** std::vector<int> *ids*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<int>> &*results*`)`

void **BatchSearchByIds**(**const** std::vector<int> *ids*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<std::pair<int, float>>> &*results*)

Search k nearest items (as ids) to each query item (batch search with multi-threads).

Parameters

- `ids`: Query ids.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `n_threads`: Number of threads to use for search.
- `[out] result`: vector of `k` nearest items for each input query item in the order passed to parameter `ids`.

3.3.3 Full Reference

This is a full documentation of C++ implementation auto-generated from code comments.

CPP Reference

Class Hierarchy

File Hierarchy

Full API

Namespaces

Namespace n2

Contents

Classes

- *Class Data*
- *Class DotDistance*
- *Class FurtherFirst*
- *Template Class HeuristicNeighborSelectingPolicies*
- *Class Hnsw*
- *Class HnswBuild*
- *Template Class HnswBuildImpl*
- *Class HnswModel*
- *Class HnswNode*
- *Class HnswSearch*
- *Template Class HnswSearchImpl*
- *Class L2Distance*
- *Template Class MinHeap*
- *Class MinHeap::Item*
- *Class Mmap*
- *Class NaiveNeighborSelectingPolicies*
- *Class Utils*
- *Class VisitedList*

Enums

- *Enum DistanceKind*
- *Enum GraphPostProcessing*
- *Enum NeighborSelectingPolicy*

Typedefs

- *Typedef n2::DistanceMaxHeap*
- *Typedef n2::HnswBuildAngular*
- *Typedef n2::HnswBuildDot*
- *Typedef n2::HnswBuildL2*
- *Typedef n2::HnswSearchAngular*
- *Typedef n2::HnswSearchDot*
- *Typedef n2::HnswSearchL2*
- *Typedef n2::IdDistancePair*
- *Typedef n2::IdDistancePairMinHeap*

Classes and Structs

Struct IdDistancePairMaxHeapComparer

- Defined in file_include_n2_max_heap.h

Struct Documentation

struct n2::**IdDistancePairMaxHeapComparer**

Public Functions

bool **operator()**(**const** *IdDistancePair* &*p1*, **const** *IdDistancePair* &*p2*) **const**

Struct IdDistancePairMinHeapComparer

- Defined in file_include_n2_min_heap.h

Struct Documentation

struct n2::**IdDistancePairMinHeapComparer**

Public Functions

bool **operator()**(**const** *IdDistancePair* &*p1*, **const** *IdDistancePair* &*p2*) **const**

Class AngularDistance

- Defined in file_include_n2_distance.h

Class Documentation

class n2::**AngularDistance**

Public Functions

float **operator()**(**const** float **v1*, **const** float **v2*, size_t *qty*) **const**

float **operator()**(**const** *HnswNode* **n1*, **const** *HnswNode* **n2*, size_t *qty*) **const**

Class BaseNeighborSelectingPolicies

- Defined in file_include_n2_heuristic.h

Inheritance Relationships

Derived Types

- `public n2::HeuristicNeighborSelectingPolicies< DistFuncType >` (*Template Class HeuristicNeighborSelectingPolicies*)
- `public n2::NaiveNeighborSelectingPolicies` (*Class NaiveNeighborSelectingPolicies*)

Class Documentation

class `n2::`**BaseNeighborSelectingPolicies**

Subclassed by *n2::HeuristicNeighborSelectingPolicies< DistFuncType >*, *n2::NaiveNeighborSelectingPolicies*

Public Functions

BaseNeighborSelectingPolicies()

~BaseNeighborSelectingPolicies() = 0

void **Select**(size_t *m*, size_t *dim*, bool *select_nn*, std::priority_queue<*FurtherFirst*> &*result*) = 0

Class CloserFirst

- Defined in file_include_n2_sort.h

Class Documentation

class `n2::`**CloserFirst**

Public Functions

CloserFirst(*HnswNode* **node*, float *distance*)

float **GetDistance**() **const**

HnswNode ***GetNode**() **const**

bool **operator<**(**const** *CloserFirst* &*n*) **const**

Class Data

- Defined in file_include_n2_data.h

Class Documentation

class n2::**Data**

Public Functions

Data(**const** std::vector<float> &*data*)

const std::vector<float> &**GetData**() **const**

const float ***GetRawData**() **const**

Class DotDistance

- Defined in file_include_n2_distance.h

Class Documentation

class n2::**DotDistance**

Public Functions

float **operator()**(**const** float **v1*, **const** float **v2*, size_t *qty*) **const**

float **operator()**(**const** *HnswNode* **n1*, **const** *HnswNode* **n2*, size_t *qty*) **const**

Class FurtherFirst

- Defined in file_include_n2_sort.h

Class Documentation

class n2::**FurtherFirst**

Public Functions

FurtherFirst(*HnswNode* **node*, float *distance*)

float **GetDistance**() **const**

HnswNode ***GetNode**() **const**

bool **operator<**(**const** *FurtherFirst* &*n*) **const**

Template Class HeuristicNeighborSelectingPolicies

- Defined in file_include_n2_heuristic.h

Inheritance Relationships

Base Type

- `public n2::BaseNeighborSelectingPolicies` (*Class BaseNeighborSelectingPolicies*)

Class Documentation

template<typename **DistFuncType**>
class `n2::` **HeuristicNeighborSelectingPolicies** : **public** n2::*BaseNeighborSelectingPolicies*

Public Functions

HeuristicNeighborSelectingPolicies()

HeuristicNeighborSelectingPolicies(bool *save_remain*)

~HeuristicNeighborSelectingPolicies() **override**

void **Select**(size_t *m*, size_t *dim*, bool *select_nn*, std::priority_queue<*FurtherFirst*> &*result*) **override**

Returns selected neighbors to result (analagous to SELECT-NEIGHBORS-HEURISTIC in Yu. A. Malkov's paper.)

select_nn: if true, select 0.25 * m nearest neighbors to result without applying the heuristic algorithm

Class Hnsw

- Defined in file_include_n2_hnsw.h

Class Documentation

class `n2::` **Hnsw**

Public Functions

Hnsw()

Hnsw(int *dim*, std::string *metric* = "angular")

Makes an instance of *Hnsw* Index.

Return A new *Hnsw* index.

Parameters

- `dim`: Dimension of vectors.

- `metric`: An optional parameter to choose a distance metric. ('angular' | 'L2' | 'dot') (default: 'angular').

Hnsw(**const** *Hnsw* &*other*)

Hnsw(*Hnsw* &&*other*) **noexcept**

~Hnsw()

Hnsw &**operator=**(**const** *Hnsw* &*other*)

Hnsw &**operator=**(*Hnsw* &&*other*) **noexcept**

void **AddData**(**const** std::vector<float> &*data*)
Adds vector to *Hnsw* index.

Parameters

- `data`: A vector with dimension `dim`.

void **SetConfigs**(**const** std::vector<std::pair<std::string, std::string>> &*configs*)
Set configurations by key/value pairs.

To set configurations as default values, pass negative values to configuration parameters.

void **Build**(int *m* = -1, int *max_m0* = -1, int *ef_construction* = -1, int *n_threads* = -1, float *mult* = -1, *NeighborSelectingPolicy* *neighbor_selecting* = *NeighborSelectingPolicy*::*HEURISTIC*, *GraphPostProcessing* *graph_merging* = *GraphPostProcessing*::*SKIP*, bool *ensure_k* = false)
Builds a hnsw graph with given configurations.

To see other available values for `neighbor_selecting` and `graph_merging`, refer to *NeighborSelectingPolicy()* and *GraphPostProcessing()*.

See *Fit()*, *SetConfigs()*

Parameters

- `m`: Max number of edges for nodes at level > 0 (default: 12).
- `max_m0`: Max number of edges for nodes at level == 0 (default: 24).
- `ef_construction`: Refer to HNSW paper for its role (default: 150).
- `n_threads`: Number of threads for building index.
- `mult`: Level multiplier (default value recommended) (default: 1/log(1.0*M)).
- `NeighborSelectingPolicy`: Neighbor selecting policy.
- `GraphPostProcessing`: Graph merging heuristic.

void **Fit**()
Builds a hnsw graph with given configurations.

bool **SaveModel**(**const** std::string &*fname*) **const**
Saves the index to disk.

Parameters

- `fname`: An index file name.

bool **LoadModel**(**const** std::string &*fname*, **const** bool *use_mmap* = true)
Loads an index from disk.

Parameters

- `fname`: An index file name.
- `use_mmap`: An optional parameter (default: true). If this parameter is set, N2 loads model through mmap.

void **`UnloadModel`**`()`
Unloads the loaded index file.

void **`SearchByVector`**`(`**`const`** std::vector<float> &*qvec*, size_t *k*, size_t *ef_search*, std::vector<int> &*result*`)`

void **`SearchByVector`**`(`**`const`** std::vector<float> &*qvec*, size_t *k*, size_t *ef_search*, std::vector<std::pair<int, float>> &*result*`)`
Search k nearest items (as vectors) to a query item.

Parameters

- `qvec`: A query vector.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `[out] result`: `k` nearest items.

void **`SearchById`**`(`int *id*, size_t *k*, size_t *ef_search*, std::vector<int> &*result*`)`

void **`SearchById`**`(`int *id*, size_t *k*, size_t *ef_search*, std::vector<std::pair<int, float>> &*result*`)`
Search k nearest items (as ids) to a query item.

Parameters

- `id`: A query id.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `[out] result`: `k` nearest items.

void **`BatchSearchByVectors`**`(`**`const`** std::vector<std::vector<float>> &*qvecs*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<int>> &*results*`)`

void **`BatchSearchByVectors`**`(`**`const`** std::vector<std::vector<float>> &*qvecs*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<std::pair<int, float>>> &*results*`)`
Search k nearest items (as vectors) to each query item (batch search with multi-threads).

Parameters

- `qvecs`: Query vectors.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `n_threads`: Number of threads to use for search.

- `[out] result`: vector of `k` nearest items for each input query item in the order passed to parameter `qvecs`.

void **BatchSearchByIds**(**const** std::vector<int> *ids*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<int>> &*results*)

void **BatchSearchByIds**(**const** std::vector<int> *ids*, size_t *k*, size_t *ef_search*, size_t *n_threads*, std::vector<std::vector<std::pair<int, float>>> &*results*)

Search k nearest items (as ids) to each query item (batch search with multi-threads).

Parameters

- `ids`: Query ids.
- `k`: k value.
- `ef_search`: (default: 50 * k). If you pass a negative value to ef_search, ef_search will be set as the default value.
- `n_threads`: Number of threads to use for search.
- `[out] result`: vector of `k` nearest items for each input query item in the order passed to parameter `ids`.

void **PrintDegreeDist**() **const**

Prints degree distributions.

void **PrintConfigs**() **const**

Prints index configurations.

Class HnswBuild

- Defined in file_include_n2_hnsw_build.h

Inheritance Relationships

Derived Type

- `public n2::HnswBuildImpl< DistFuncType >` (*Template Class HnswBuildImpl*)

Class Documentation

class `n2::`**HnswBuild**

Subclassed by *n2::HnswBuildImpl< DistFuncType >*

Public Functions

HnswBuild(int *dim*, *DistanceKind* *metric*)

~HnswBuild()

HnswBuild(**const** *HnswBuild*&) = delete

void **operator=**(**const** *HnswBuild*&) = delete

void **AddData**(**const** std::vector<float> &*data*)

void **SetConfigs**(**const** std::vector<std::pair<std::string, std::string>> &*configs*)

std::shared_ptr<**const** *HnswModel*> **Build**(int *m*, int *max_m0*, int *ef_construction*, int *n_threads*, float *mult*, *NeighborSelectingPolicy* *neighbor_selecting*, *GraphPostProcessing* *graph_merging*)

std::shared_ptr<**const** *HnswModel*> **Build**()

void **PrintDegreeDist**() **const**

void **PrintConfigs**() **const**

Public Static Functions

std::unique_ptr<*HnswBuild*> **GenerateBuilder**(int *dim*, *DistanceKind* *metric*)

Protected Functions

void **SetConfigs**(int *m*, int *max_m0*, int *ef_construction*, int *n_threads*, float *mult*, *NeighborSelectingPolicy* *neighbor_selecting*, *GraphPostProcessing* *graph_merging*)

int **GetRandomNodeLevel**()

int **GetRandomSeedPerThread**()

void **BuildGraph**(bool *reverse*)

void **InitPolicies**() = 0

void **InsertNode**(*HnswNode* *qnode*, *VisitedList* *visited_list*) = 0

void **SearchAtLayer**(*HnswNode* *qnode*, **const** std::vector<*HnswNode**> &*enterpoints*, int *level*, *VisitedList* *visited_list*, std::priority_queue<*FurtherFirst*> &*result*) = 0

void **Link**(*HnswNode* *source*, *HnswNode* *target*, int *level*) = 0

void **MergeEdgesOfTwoGraphs**(**const** std::vector<*HnswNode**> &*another_nodes*) = 0

Protected Attributes

std::shared_ptr<spdlog::logger> **logger_**

const std::string **n2_signature** = "TOROS_N2@N9R4"

size_t **m_** = 12

size_t **max_m_** = 12

size_t **max_m0_** = 24

size_t **ef_construction_** = 150

float **level_mult_** = 1 / log(1.0 * *m_*)

int **n_threads_** = 1

NeighborSelectingPolicy **neighbor_selecting_** = *NeighborSelectingPolicy*::*HEURISTIC*

NeighborSelectingPolicy **post_neighbor_selecting_** = *NeighborSelectingPolicy*::*HEURISTIC_SAVE_REMAINS*

GraphPostProcessing **post_graph_process_** = *GraphPostProcessing*::*SKIP*

int **max_level_** = 0

HnswNode ***enterpoint_** = nullptr

std::vector<*Data*> **data_list_**

std::vector<*HnswNode**> **nodes_**

int **num_nodes_** = 0

size_t **data_dim_** = 0

DistanceKind **metric_**

std::mutex **max_level_guard_**

Template Class HnswBuildImpl

- Defined in file_include_n2_hnsw_build.h

Inheritance Relationships

Base Type

- `public n2::HnswBuild` (*Class HnswBuild*)

Class Documentation

template<typename **DistFuncType**>
class `n2::` **HnswBuildImpl** : **public** n2::*HnswBuild*

Public Functions

HnswBuildImpl(int *dim*, *DistanceKind* *metric*)

~HnswBuildImpl() **override**

Protected Functions

void **InitPolicies**() **override**

void **InsertNode**(*HnswNode* *qnode, *VisitedList* *visited_list) **override**

void **SearchAtLayer**(*HnswNode* *qnode, **const** std::vector<*HnswNode**> &*enterpoint*, int *level*, *VisitedList* *visited_list, std::priority_queue<*FurtherFirst*> &*result*) **override**

void **Link**(*HnswNode* *source, *HnswNode* *target, int *level*) **override**

void **MergeEdgesOfTwoGraphs**(**const** std::vector<*HnswNode**> &*another_nodes*) **override**

Protected Attributes

bool **is_naive_** = false

std::unique_ptr<*BaseNeighborSelectingPolicies*> **selecting_policy_**

std::unique_ptr<*BaseNeighborSelectingPolicies*> **post_selecting_policy_**

DistFuncType **dist_func_**

Class HnswModel

- Defined in file_include_n2_hnsw_model.h

Class Documentation

class n2::**HnswModel**

Public Functions

~HnswModel()

bool **SaveModelToFile**(**const** std::string &*fname*) **const**

HnswModel(**const** *HnswModel*&) = delete

void **operator=**(**const** *HnswModel*&) = delete

int **GetNumNodes**() **const**

int **GetEnterpointId**() **const**

int **GetMaxLevel**() **const**

int **GetDataDim**() **const**

DistanceKind **GetMetric**() **const**

const float ***GetData**(int *node_id*) **const**

const int ***GetHigherLevelFriendsWithSize**(int *node_id*, int *level*) **const**

const int ***GetLevel0FriendsWithSize**(int *node_id*) **const**

Public Members

int **enterpoint_id_**

int **num_nodes_**

int **max_level_**

size_t **data_dim_** = 0

DistanceKind **metric_**

char ***model_** = nullptr

uint64_t **model_byte_size_**

char ***model_higher_level_** = nullptr

char ***model_level0_** = nullptr

char ***model_level0_node_base_offset_** = nullptr

uint64_t **memory_per_data_**

uint64_t **memory_per_link_level0_**

uint64_t **memory_per_node_level0_**

uint64_t **memory_per_node_higher_level_**

Mmap ***model_mmap_** = nullptr

Public Static Functions

std::shared_ptr<**const** *HnswModel*> **GenerateModel**(**const** std::vector<*HnswNode**> *nodes*, int *enterpoint_id*, int *max_m*, int *max_m0*, *DistanceKind* *metric*, int *max_level*, size_t *data_dim*)

std::shared_ptr<**const** *HnswModel*> **LoadModelFromFile**(**const** std::string &*fname*, **const** bool *use_mmap* = true)

Class HnswNode

- Defined in file_include_n2_hnsw_node.h

Class Documentation

class n2::**HnswNode**

Public Functions

HnswNode(int *id*, **const** *Data* *data, int *level*, size_t *max_m*, size_t *max_m0*)

void **CopyHigherLevelLinksToOptIndex**(char *mem_offset, uint64_t *memory_per_node_higher_level*) **const**

void **CopyDataAndLevel0LinksToOptIndex**(char *mem_offset, int *higher_level_offset*) **const**

int **GetId**() **const**

int **GetLevel**() **const**

size_t **GetMaxM**() **const**

size_t **GetMaxM0**() **const**

const float *__GetData**() **const**

std::vector<*HnswNode**> &**GetFriends**(int *level*)

void **SetFriends**(int *level*, std::vector<*HnswNode**> &*new_friends*)

std::mutex &**GetAccessGuard**()

Class HnswSearch

- Defined in file_include_n2_hnsw_search.h

Inheritance Relationships

Derived Type

- `public n2::HnswSearchImpl< DistFuncType >` (*Template Class HnswSearchImpl*)

Class Documentation

class `n2::`**HnswSearch**

Subclassed by *n2::HnswSearchImpl< DistFuncType >*

Public Functions

~HnswSearch()

void **SearchByVector**(**const** std::vector<float> &*qvec*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<int> &*result*) = 0

void **SearchByVector**(**const** std::vector<float> &*qvec*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<std::pair<int, float>> &*result*) = 0

void **SearchById**(int *id*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<int> &*result*) = 0

void **SearchById**(int *id*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<std::pair<int, float>> &*result*) = 0

Public Static Functions

std::unique_ptr<*HnswSearch*> **GenerateSearcher**(std::shared_ptr<**const** *HnswModel*> *model*, size_t *data_dim*, *DistanceKind* *metric*)

Template Class HnswSearchImpl

- Defined in file_include_n2_hnsw_search_impl.h

Inheritance Relationships

Base Type

- `public n2::HnswSearch` (*Class HnswSearch*)

Class Documentation

template<typename **DistFuncType**>
class `n2::` **HnswSearchImpl** : **public** n2::*HnswSearch*

Public Functions

HnswSearchImpl(std::shared_ptr<**const** *HnswModel*> *model*, size_t *data_dim*, *DistanceKind* *metric*)

void **SearchByVector**(**const** std::vector<float> &*qvec*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<int> &*result*) **override**

void **SearchByVector**(**const** std::vector<float> &*qvec*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<std::pair<int, float>> &*result*) **override**

void **SearchById**(int *id*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<int> &*result*) **override**

void **SearchById**(int *id*, size_t *k*, int *ef_search*, bool *ensure_k*, std::vector<std::pair<int, float>> &*result*) **override**

Protected Functions

template<typename **ResultType**>
void **SearchByVector_**(**const** std::vector<float> &*qvec*, size_t *k*, int *ef_search*, bool *ensure_k*, *ResultType* &*result*)

void **CallSearchById_**(int *cur_node_id*, float *cur_dist*, **const** float **qraw*, size_t *k*, size_t *ef_search*, bool *ensure_k*, std::vector<int> &*result*)

void **CallSearchById_**(int *cur_node_id*, float *cur_dist*, **const** float **qraw*, size_t *k*, size_t *ef_search*, bool *ensure_k*, std::vector<std::pair<int, float>> &*result*)

template<typename **ResultType**>
void **SearchById_**(int *cur_node_id*, float *cur_dist*, **const** float **qraw*, size_t *k*, size_t *ef_search*, bool *ensure_k*, *ResultType* &*result*)

template<typename **ResultType**>
void **SearchByIdV1_**(int *cur_node_id*, float *cur_dist*, **const** float **qraw*, size_t *k*, size_t *ef_search*, bool *ensure_k*, *ResultType* &*result*)

template<typename **ResultType**>
void **SearchByIdV2_**(int *cur_node_id*, float *cur_dist*, **const** float **qraw*, size_t *k*, size_t *ef_search*, bool *ensure_k*, *ResultType* &*result*)

bool **PrepareEnsureKSearch**(int *cur_node_id*, std::vector<int> &*result*, *IdDistancePairMinHeap* &*visited_nodes*)

bool **PrepareEnsureKSearch**(int *cur_node_id*, std::vector<std::pair<int, float>> &*result*, *IdDistancePairMinHeap* &*visited_nodes*)

void **MakeSearchResult**(size_t *k*, *IdDistancePairMinHeap* &*candidates*, *IdDistancePairMinHeap* &*visited_nodes*, std::vector<int> &*result*)

void **MakeSearchResult**(size_t *k*, *IdDistancePairMinHeap* &*candidates*, *IdDistancePairMinHeap* &*visited_nodes*, std::vector<std::pair<int, float>> &*result*)

Protected Attributes

std::shared_ptr<**const** *HnswModel*> **model_**

std::unique_ptr<*VisitedList*> **visited_list_**

size_t **data_dim_**

DistanceKind **metric_**

DistFuncType **dist_func_**

std::vector<float> **normalized_vec_**

std::vector<std::pair<int, float>> **ensure_k_path_**

char ***model_higher_level_** = nullptr

char ***model_level0_** = nullptr

char ***model_level0_node_base_offset_** = nullptr

uint64_t **memory_per_node_level0_**

uint64_t **memory_per_node_higher_level_**

Class L2Distance

- Defined in file_include_n2_distance.h

Class Documentation

class n2::**L2Distance**

Public Functions

float **operator()**(**const** float *v1, **const** float *v2, size_t *qty*) **const**

float **operator()**(**const** *HnswNode* *n1, **const** *HnswNode* *n2, size_t *qty*) **const**

Template Class MinHeap

- Defined in file_include_n2_min_heap.h

Nested Relationships

Nested Types

- *Class MinHeap::Item*

Class Documentation

template<typename **KeyType**, typename **DataType**>
class n2::**MinHeap**

Public Functions

MinHeap()

const *KeyType* **top_key**()

Item **top**()

void **pop**()

void **push**(**const** *KeyType* &*key*, **const** *DataType* &*data*)

size_t **size**()

class Item

Public Functions

Item()

Item(**const** *KeyType* &*key*)

Item(**const** *KeyType* &*key*, **const** *DataType* &*data*)

bool **operator<**(**const** *Item* &*i2*) **const**

Public Members

KeyType **key**

DataType **data**

Class MinHeap::Item

- Defined in file_include_n2_min_heap.h

Nested Relationships

This class is a nested type of *Template Class MinHeap*.

Class Documentation

class n2::*MinHeap*::**Item**

Public Functions

Item()

Item(**const** KeyType &*key*)

Item(**const** KeyType &*key*, **const** DataType &*data*)

bool **operator<**(**const** *Item* &*i2*) **const**

Public Members

KeyType **key**

DataType **data**

Class Mmap

- Defined in file_include_n2_mmap.h

Class Documentation

class n2::**Mmap**

Public Functions

Mmap(char **const** **fname*)

~Mmap()

void **Map**(char **const** **fname*)

void **UnMap**()

size_t **QueryFileSize**() **const**

char ***GetData**() **const**

bool **IsOpen**() **const**

int **GetFileHandle**() **const**

size_t **GetFileSize**() **const**

Class NaiveNeighborSelectingPolicies

- Defined in file_include_n2_heuristic.h

Inheritance Relationships

Base Type

- `public n2::BaseNeighborSelectingPolicies` (*Class BaseNeighborSelectingPolicies*)

Class Documentation

class `n2::`**NaiveNeighborSelectingPolicies** : **public** n2::*BaseNeighborSelectingPolicies*

Public Functions

NaiveNeighborSelectingPolicies()

~NaiveNeighborSelectingPolicies() **override**

void **Select**(size_t *m*, size_t *dim*, bool *select_nn*, std::priority_queue<*FurtherFirst*> &*result*) **override**

Class Utils

- Defined in file_include_n2_utils.h

Class Documentation

class n2::**Utils**

Public Static Functions

void **NormalizeVector**(**const** std::vector<float> &*in*, std::vector<float> &*out*)

Class VisitedList

- Defined in file_include_n2_visited_list.h

Class Documentation

class n2::**VisitedList**

Public Functions

VisitedList(int *size*)

~VisitedList()

bool **Visited**(unsigned int *index*) **const**

bool **NotVisited**(unsigned int *index*) **const**

void **MarkAsVisited**(unsigned int *index*)

void **Reset**()

unsigned int ***GetVisited**()

unsigned int **GetVisitMark**()

Enums

Enum DistanceKind

- Defined in file_include_n2_common.h

Enum Documentation

enum n2::**DistanceKind**

Values:

enumerator UNKNOWN = -1

enumerator ANGULAR = 0

enumerator L2 = 1

enumerator DOT = 2

Enum GraphPostProcessing

- Defined in file_include_n2_common.h

Enum Documentation

enum n2::**GraphPostProcessing**
Graph merging heuristic.

Values:

enumerator SKIP = 0
Do not merge (recommended for large-scale data (over 10M)).

enumerator MERGE_LEVEL0 = 1
Performs an additional graph build in reverse order, then merges edges at level 0. So, it takes twice the build time compared to "skip" but shows slightly higher accuracy. (recommended for data under 10M scale).

Enum NeighborSelectingPolicy

- Defined in file_include_n2_common.h

Enum Documentation

enum n2::**NeighborSelectingPolicy**
Neighbor selecting policy.

Values:

enumerator NAIVE = 0
Select closest neighbors (not recommended).

enumerator HEURISTIC = 1
Select neighbors using algorithm4 on HNSW paper (recommended).

enumerator HEURISTIC_SAVE_REMAINS = 2
Experimental.

Typedefs

Typedef n2::DistanceMaxHeap

- Defined in file_include_n2_max_heap.h

Typedef Documentation

typedef boost::heap::d_ary_heap<float, boost::heap::arity<4>> n2::**DistanceMaxHeap**

Typedef n2::HnswBuildAngular

- Defined in file_include_n2_hnsw_build.h

Typedef Documentation

using n2::**HnswBuildAngular** = *HnswBuildImpl*<*AngularDistance*>

Typedef n2::HnswBuildDot

- Defined in file_include_n2_hnsw_build.h

Typedef Documentation

using n2::**HnswBuildDot** = *HnswBuildImpl*<*DotDistance*>

Typedef n2::HnswBuildL2

- Defined in file_include_n2_hnsw_build.h

Typedef Documentation

using n2::**HnswBuildL2** = *HnswBuildImpl*<*L2Distance*>

Typedef n2::HnswSearchAngular

- Defined in file_include_n2_hnsw_search_impl.h

Typedef Documentation

using n2::**HnswSearchAngular** = *HnswSearchImpl*<*AngularDistance*>

Typedef n2::HnswSearchDot

- Defined in file_include_n2_hnsw_search_impl.h

Typedef Documentation

using n2::**HnswSearchDot** = *HnswSearchImpl*<*DotDistance*>

Typedef n2::HnswSearchL2

- Defined in file_include_n2_hnsw_search_impl.h

Typedef Documentation

using n2::**HnswSearchL2** = *HnswSearchImpl*<*L2Distance*>

Typedef n2::IdDistancePair

- Defined in file_include_n2_max_heap.h

Typedef Documentation

typedef std::pair<int, float> n2::**IdDistancePair**

Typedef n2::IdDistancePairMinHeap

- Defined in file_include_n2_min_heap.h

Typedef Documentation

typedef boost::heap::d_ary_heap<*IdDistancePair*, boost::heap::arity<4>, boost::heap::compare<*IdDistancePairMinHeapComparer*

3.4 Go Interface

3.4.1 Basic Usage

```
package main

import (
    "n2"
    "fmt"
    "math/rand"
)
```

(continues on next page)

(continued from previous page)

```
func main() {
     f := 3
     t := n2.NewHnswIndex(f)
     for i := 0; i < 1000; i++ {
       item := make([]float32, 0, f)
       for x:= 0; x < f; x++ {
           item = append(item, rand.Float32())
       }
       t.AddData(item)
     }
     t.Build(5, 10, 4, 10, 3.5, "heuristic", "skip")
     t.PrintConfigs()
     t.SaveModel("test.ann")
     var result []int
     var distance []float32
     t.SearchByVector([]float32{2, 1, 0}, 1000, -1, &result, &distance)
     fmt.Println(result)
     fmt.Println(distance)
}
```

You can see more code examples at examples/go.

3.4.2 Main Interface

HnswIndex(dim, metric)

- Returns a new Hnsw index.
- `dim` (int): Dimension of vectors.
- `metric` (string): An optional parameter to choose a distance metric ('angular' | 'L2' | 'dot').

index.AddData(v)

- Adds vector `v`.
- `v` (list of float): A vector with dimension `dim`.

index.Build(M, Max_M0, ef_construction, n_threads, mult, neighbor_selecting, graph_merging)

- Builds a hnsw graph with given configurations.
- `M` (int): Max number of edges for nodes at level > 0 (default: 12).
- `Max_M0` (int): Max number of edges for nodes at level == 0 (default: 24).
- `ef_construction` (int): efConstruction (see HNSW paper.) (default: 100).
- `n_threads` (int): Number of threads for building index.
- `mult` (float): Level multiplier (recommended: use default value) (default: 1/log(1.0*M)).
- `neighbor_selecting` (string): Neighbor selecting policy.
  - Available values
    * `"heuristic"` (default): Select neighbors using algorithm4 on HNSW paper (recommended).

        * "naive": Select closest neighbors (not recommended).
- graph_merging (string): Graph merging heuristic.
  - Available values
    * "skip" (default): Do not merge (recommended for large-scale data (over 10M)).
    * "merge_level0": Performs an additional graph build in reverse order, then merges edges at level 0. So, it takes twice the build time compared to "skip" but shows slightly higher accuracy. (recommended for data under 10M scale).

index.SaveModel(fname)

- Saves the index to disk.
- fname (string)

index.LoadModel(fname, use_mmap)

- Loads an index from disk with mmap.
- fname (string)
- use_mmap (bool): An optional parameter (default: true). If this parameter is set, N2 loads model through mmap.

index.UnloadModel()

- Unloads (unmap) the index.

index.SearchByVector(item_id, k, ef_search=-1, vectors, distances)

- Returns k nearest items (as vectors) to a query item.
- ef_search (int): (default: 50 * k).

index.SearchById(v, k, ef_search=-1, vectors, distances)

- Returns k nearest items (as ids) to a query item.
- v (list of float): A query vector.
- k (int)
- ef_search (int): (default: 50 * k).

Note: Currently, batch search functions are not supported in Go binding.

3.5 N2 Benchmark

This page is a detailed explanation of how we performed benchmark experiments.

You can also see benchmarks of ANN libraries in Python at ann-benchmarks.com. Note that N2 version 0.1.6 is used in ann-benchmarks.com (last checked on October 8th, 2020) and we are continuing our efforts to improve N2 performance.

3.5.1 Benchmark Focus

These are some factors that we focus on when developing N2.

1. Our ANN algorithm should run fast even when dealing with large-scale datasets.
2. Our ANN algorithm should minimize the time required to build an index file - in order to be applied to real-world scenario where dataset changes frequently (e.g. create/update/delete), such as in online content services like news portal.

Therefore, our main criteria for benchmark are set as below:

- How long does it take to build the index file?
- How long does it take to get results from the large dataset?
- How large memory does it take to run large dataset?

3.5.2 Test Dataset

Dataset Description

To test large amounts of data, we use `youtube` dataset that contains 14520986 samples, where each sample has 40 data points.

feature1(float32)	feature2(float32)	.....	feature2(float32)	feature40(float32)
-0.167898	0.160478	.....	0.104421	0.0503584

How to Download the Benchmark Dataset

You can download benchmark dataset with the script we provide in Download dataset.

We also share `youtube` dataset through google drive. It consists of two plain text files, `youtube.txt` and `youtube.txt.vids`. `youtube.txt` is a file containing the information of dataset samples and `youtube.txt.vids` is a file containing the dataset metadata information. Each line is the metadata corresponding to each sample in `youtube.txt`.

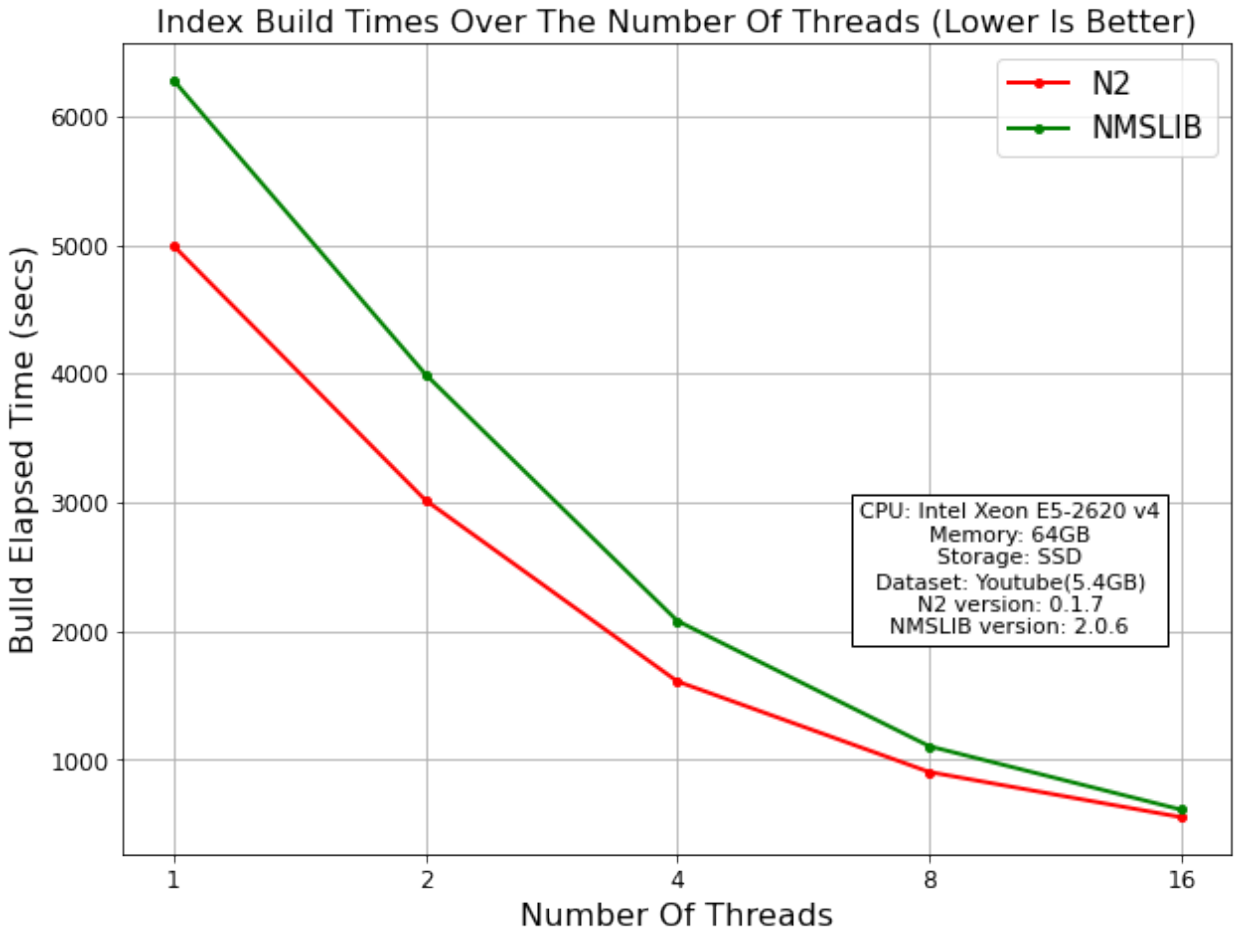
DSID	VID	Youtube link
34XnPr4YKpo8wE_mEl	Z1Jilm0TZHY	http://www.youtube.com/watch?v=Z1Jilm0TZHY

3.5.3 Test Environment

- CPU: Intel(R) Xeon(R) CPU E5-2620 v4
- Memory: 64GB
- Storage: SSD
- Dataset: Youtube(5.4GB)
- N2 version: 0.1.7
- NMSLIB version: 2.0.6
- g++ (gcc): 7.3.1

3.5.4 Index Build Time

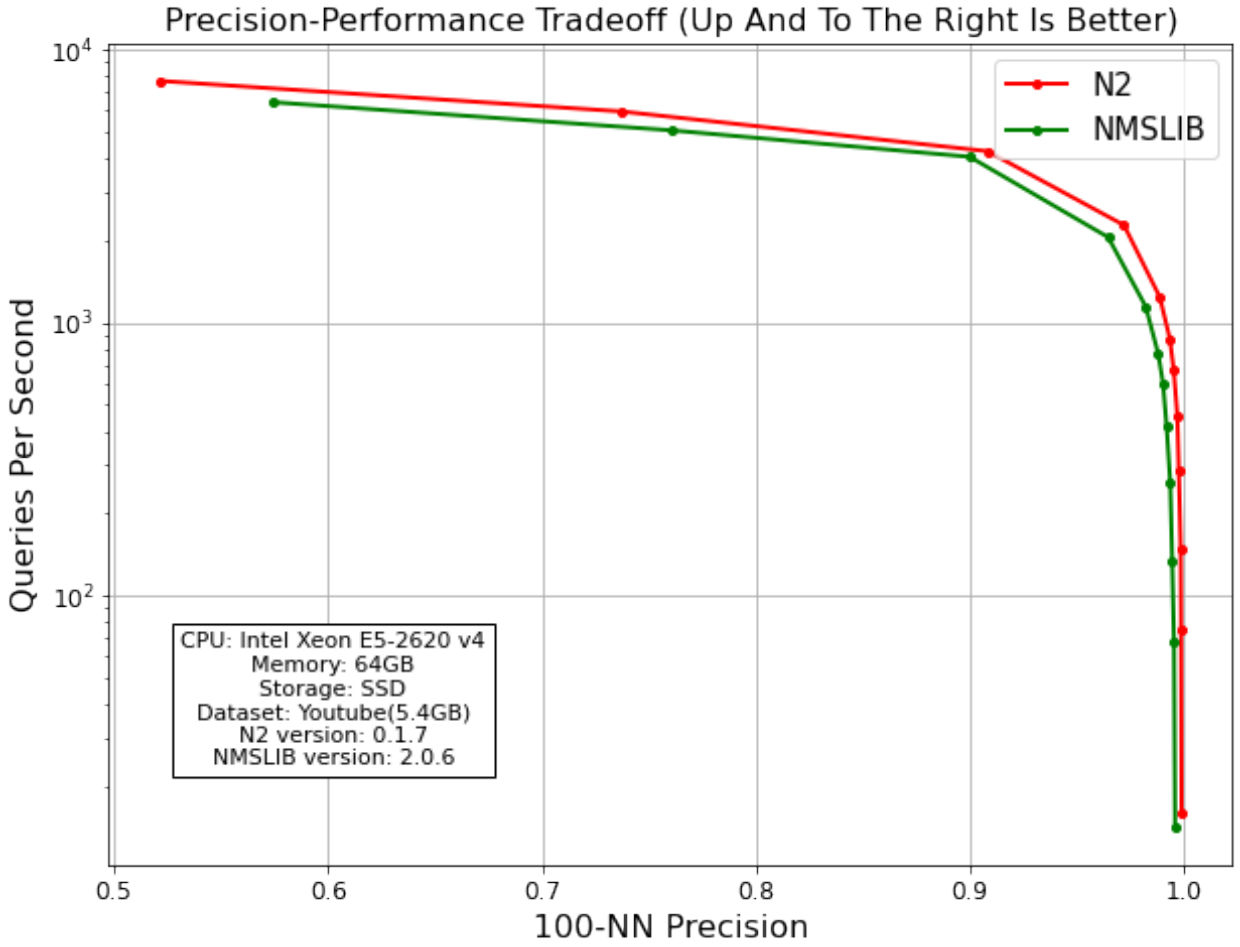
The following is a comparison of the index build times taken when using different numbers of threads. N2 builds index file 10~24% faster than NMSLIB.



Library	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
N2 (Index Size: 3.7GB)	4995.73 sec	3018.57 sec	1609.89 sec	905.87 sec	554.81 sec
NMSLIB (Index Size: 3.9GB)	6282.5 sec	3996.88 sec	2080.36 sec	1106.18 sec	613.18 sec

3.5.5 Search Speed

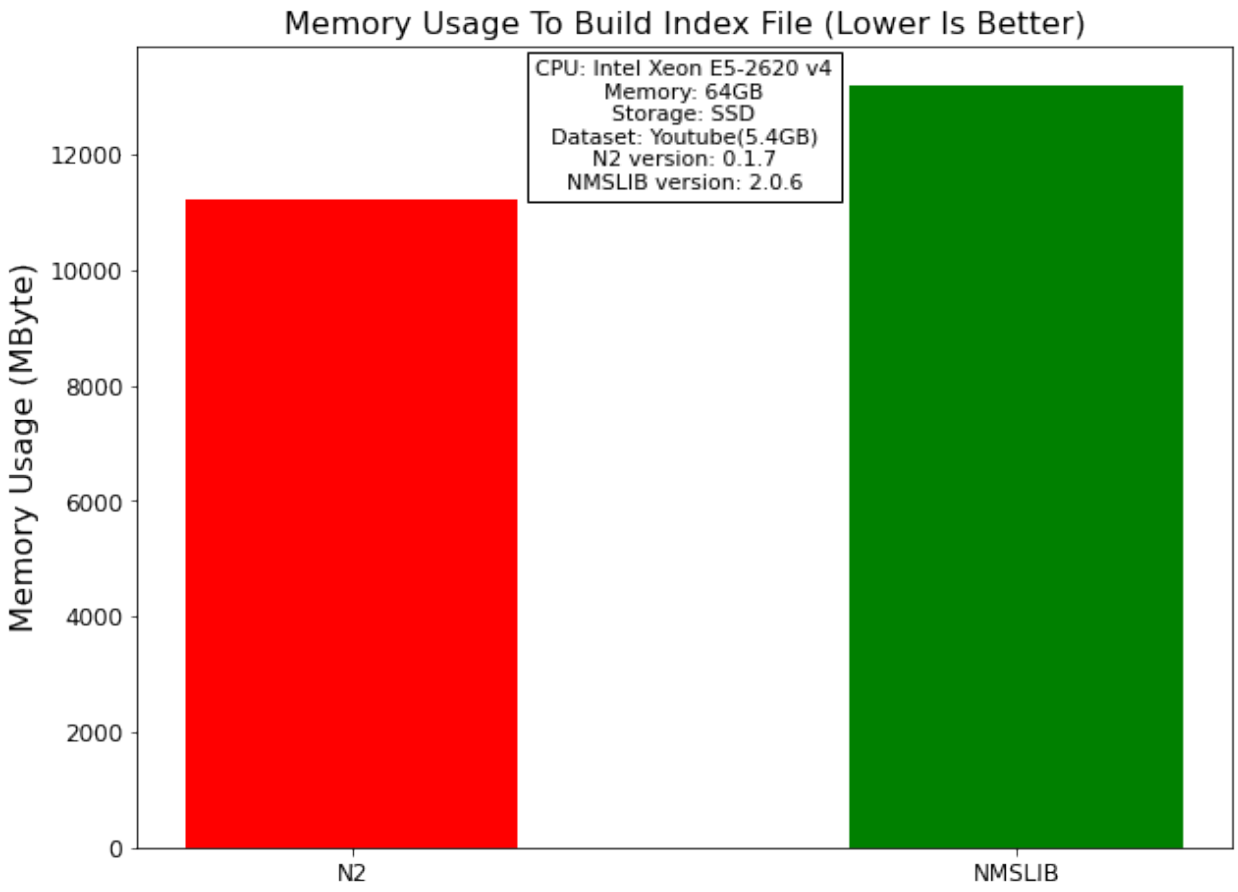
The data below shows tradeoff between QPS(Queries Per Second) and accuracy. Both N2 and NMSLIB shows similar search performance.



Parameter	Search Time (N2)	Accuracy (N2)	Search Time (NMSLIB)	Accuracy (NMSLIB)
M: 12, efCon: 100, efSearch: 25	0.000130227	0.52136	0.000155903	0.574523
M: 12, efCon: 100, efSearch: 50	0.000168451	0.736898	0.000197621	0.760703
M: 12, efCon: 100, efSearch: 100	0.000235572	0.908154	0.000247012	0.899827
M: 12, efCon: 100, efSearch: 250	0.000439563	0.971894	0.000486722	0.964502
M: 12, efCon: 100, efSearch: 500	0.000805385	0.988616	0.000871604	0.982023
M: 12, efCon: 100, efSearch: 750	0.00114534	0.993323	0.00129876	0.987889
M: 12, efCon: 100, efSearch: 1000	0.00148114	0.995105	0.00166815	0.99014
M: 12, efCon: 100, efSearch: 1500	0.00219379	0.996848	0.00241407	0.991855
M: 12, efCon: 100, efSearch: 2500	0.00348781	0.997529	0.00385025	0.993514
M: 12, efCon: 100, efSearch: 5000	0.00669571	0.99839	0.00744833	0.994425
M: 12, efCon: 100, efSearch: 10000	0.0132182	0.998577	0.014742	0.995269
M: 12, efCon: 100, efSearch: 50000	0.0627954	0.998814	0.0706788	0.995788

3.5.6 Memory Usage

The data below shows the amount of memory used to build the index file, which is measured as the difference between memory usage before and after building the index file. N2 uses 15% less memory than NMSLIB.



Library	Memory Usage
N2	11222.48 MB
NMSLIB	13212.76 MB

3.5.7 Conclusion

N2 builds index file faster and uses less memory than NMSLIB, while having a similar search speed performance.

The benchmark environment uses multiple threads for index builds but a single thread for searching. In a real production environment, you will need to run concurrent searches by multiple processes or multiple threads. N2 allows you to search simultaneously using multiple processes. With mmap support in N2, it works much more efficiently than other libraries, including NMSLIB.

CHAPTER

FOUR

REFERENCES

- Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," CoRR, vol. abs/1603.09320, 2016. [Online]. Available: http://arxiv.org/abs/1603.09320
- NMSLIB: https://github.com/nmslib/nmslib
- Annoy: https://github.com/spotify/annoy

CHAPTER

FIVE

LICENSE

This software is licensed under the Apache 2 license, quoted below.

INDEX

Symbols

A

B

H

L

N

S

U