Hierarchical Multi Dimensional Histograms

Release 1.0.0

AirMettle Team

November 1, 2024
Doc Version: 1.0.0

Overview

Hierarchical Multi-Dimensional Histograms (HMDH) is a software library designed for efficient data summarization, enabling rapid exploratory analysis on integer and floating-point data. HMDH (patent pending) provides fast, space- and time-efficient encoding of data characteristics across multiple orders of magnitude in a single pass.

This public version supports one-dimensional and two-dimensional data structures and includes a graphical interface for interactive exploration. It is intended for non-commercial research and evaluation purposes only (please refer to the license).

A commercial version is available with expanded capabilities: a CLI for generating HMDH from Parquet files, support for up to four dimensions, finer resolution per dimension, and advanced querying features (e.g., range queries, finding percentile of data, etc.). HMDH can process hundreds of millions to billions of data points into compact, searchable structures that typically occupy just a fraction of a megabyte, retaining precise counts and addressing at each hierarchical level! The commercial library also supports parallel generation, merging, and comparison of histograms, facilitating analysis of variations across datasets.
Some current clients are using HMDH's one to four-dimensions in their operations.

For further information, contact: hmdh@airmettle.com

Requirements

The project has been built and tested on an AWS codebuild instance running Ubuntu 22.04 with 3 GB memory and 2 vCPUs using the image aws/codebuild/standard:7.0.
The project currently runs on x86 machines.

Installation

This describes the installation process using CMake. As prerequisites, you'll need git and CMake installed.

```Python
# Clone the repository
$ git clone https://github.com/AirMettle/HMDH.git
# Go to the root of the project
$ cd HMDH
# Install dependencies using these steps
$ cd cmake
$ mkdir build
$ cd build
$ cmake ..
$ make
# Go back to the root of the project
$ cd ../..
# Build the project
$ mkdir build
$ cd build
$ cmake ..
$ make
# Install the project so it is globally available (This step will require sudo priviliges)
$ sudo make install
```

Usage

Here is an example of CMakeLists.txt to use this library with your project.

```Python
include_directories(/usr/local/include/)
link_directories(/usr/local/lib)

# Add the main.cpp file
add_executable(Simple2DExample main.cpp)

# Link the library to the executable
target_link_libraries(Simple2DExample HMDH)
```

Construct a FPHArray type of your input data. For example:

```C/C++
std::vector<double> array = ... // construct your input data as vector

// NOTE: std::vector<double> is an example. We also support float, int32 and int64.

// pass the vector to `buildFPHArray` as shown below
FPHArray array0 = buildFPHArray(array.data(),

static_cast<int>(array.size()));
```

Pass the constructed FPHArray(s) to your desired config (see API docs to learn about other configs).

Generate a 1D buffer

Here is an example to construct a 1D compact histogram buffer:

```C/C++
// array0 is of type FPHArray constructed as shown above.
std::vector<char> compactHistogram = generate_1DxF(array0);
// Persist the compact histogram data on disk.
Write(compactHistogram, "compact1D.bin");
```

Generate a 2D buffer

Here is an example to construct a 2D compact histogram buffer:

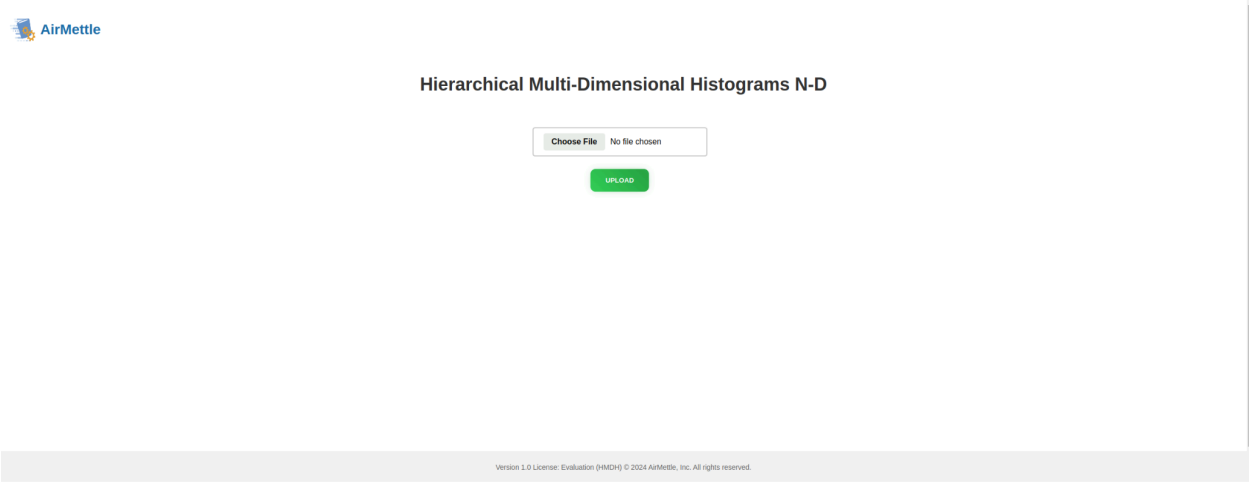
```C/C++
// array0 and array1 are of type FPHArray constructed as shown above.
std::vector<char> compactHistogram = generate_2DxF(array0, array1);
// Persist the compact histogram data on disk.
Write(compactHistogram, "compact2D.bin");
```

UI

To access the UI run the hmdh_ui application that should have been installed following the steps in Installation.

```python
$ hmdh_ui
(2024-10-09 03:02:34) [INFO    ] Crow/master server is running at http://0.0.0.0:18080 using 16 threads
(2024-10-09 03:02:34) [INFO    ] Call `app.loglevel(crow::LogLevel::Warning)` to hide Info level logs.
```

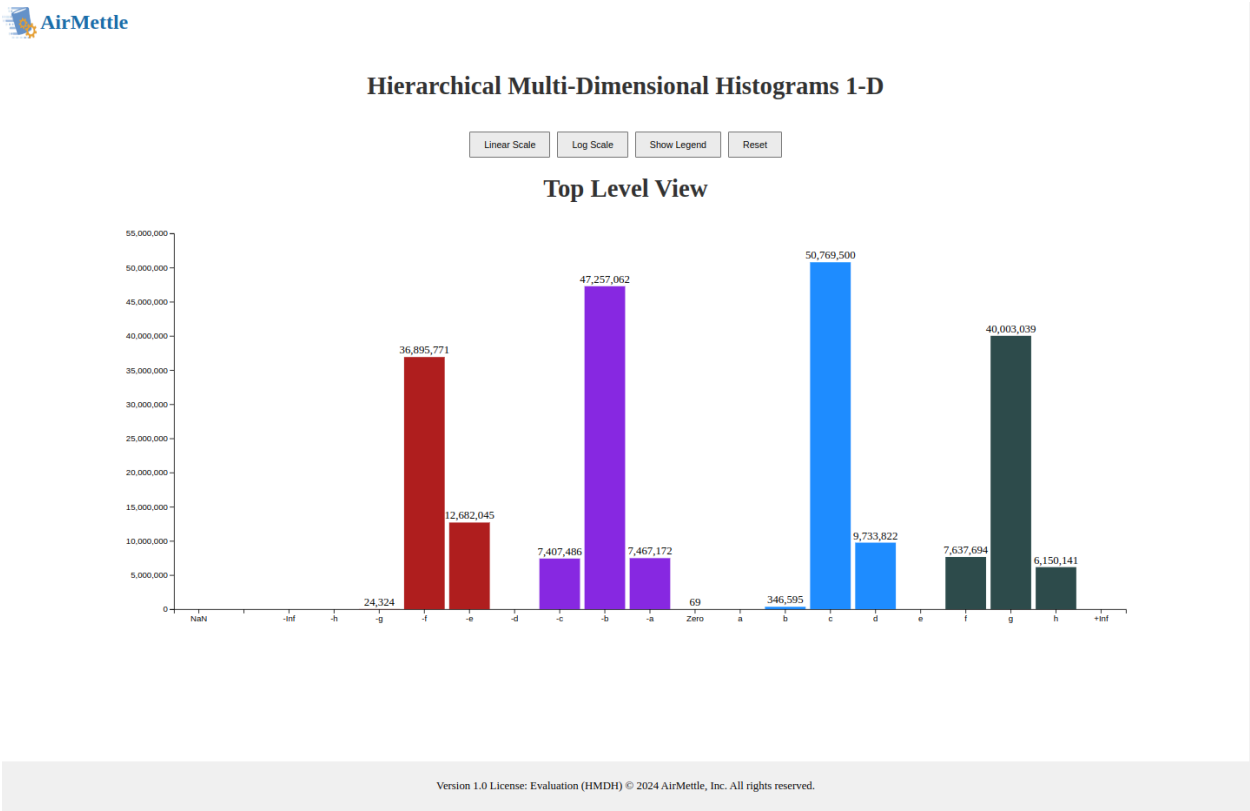
This starts a web server. Switch to your browser and go to http://localhost:18080. This should open up the landing page as shown below.



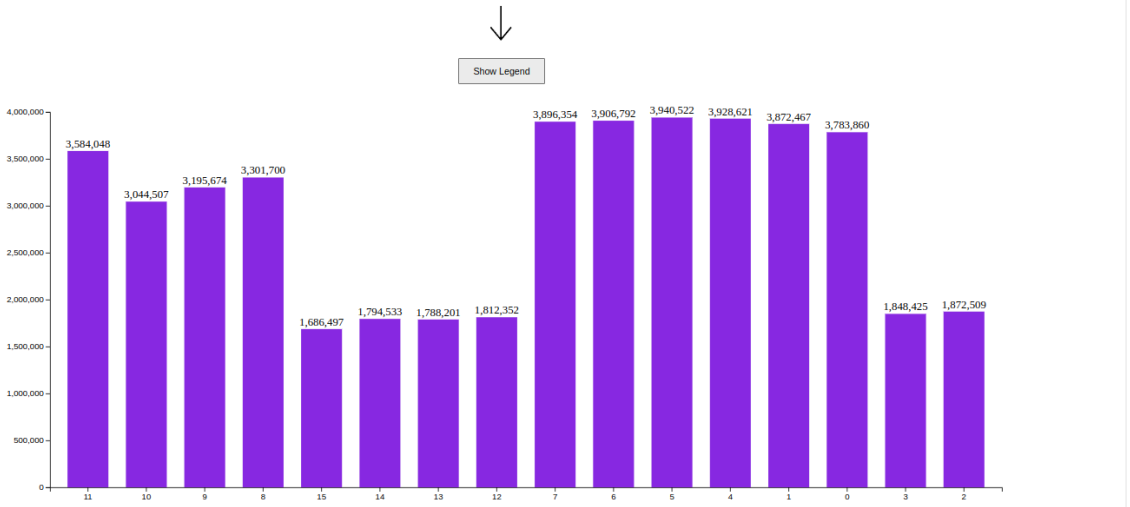
Click on Choose File and select the buffer you generated in either Generate a 1D buffer or Generate a 2D buffer step. Click on Upload to generate the plots for the buffer.

1D Plots

To generate these plots we used the JaneStreet test dataset (file id: 19JQgBJaLeHBaCV6G-Tcpqye8BQVWlqFt) from https://github.com/hipdac-lab/FCBench. The dataset contained 226+ million values with an on-disk size of 1.69GB of which we had 166+ million unique. We filed those 226+ million values in ~25K bins. The buffer we generated of this dataset using **generate_1DxF** has an on-disk size of ~71 KB uncompressed.
Using the same dataset and **generate_1DxP** (available in the commercial release) with higher precision, we filed 226+ million values in ~393K bins and an on-disk size of ~0.5 MB uncompressed.



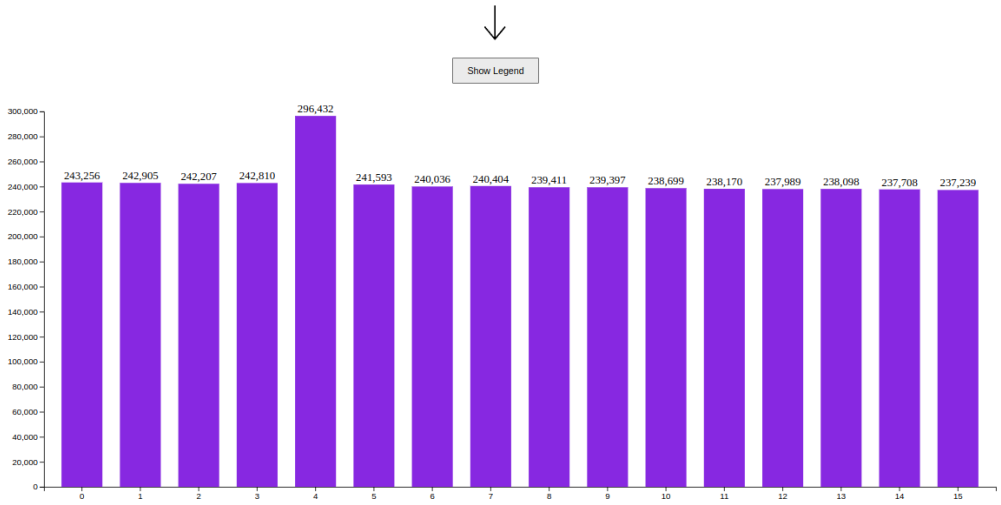
Top level classification of values



Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

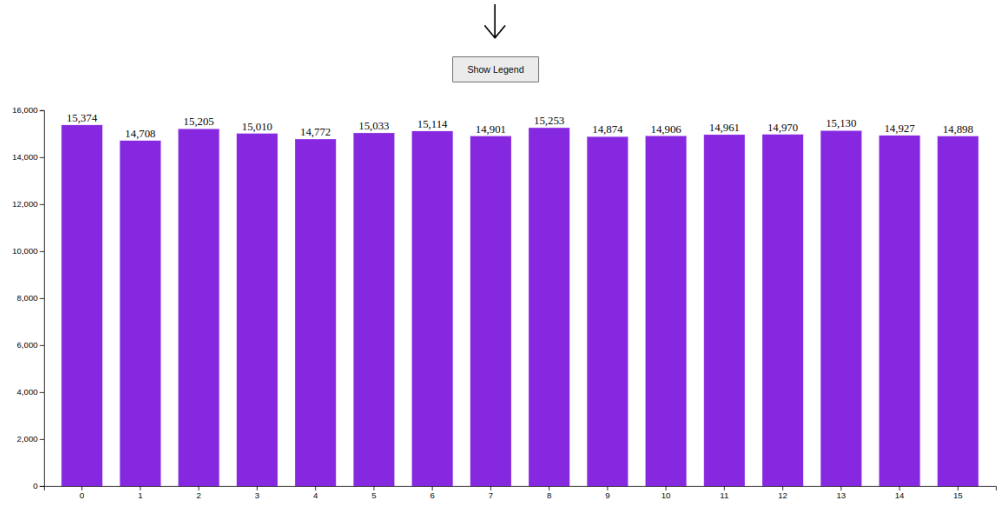
Plot for level 1

NOTE: Clicking on the ***Show Legend*** (at each level) will display the range of values under each histogram bucket.



Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Plot for level 2

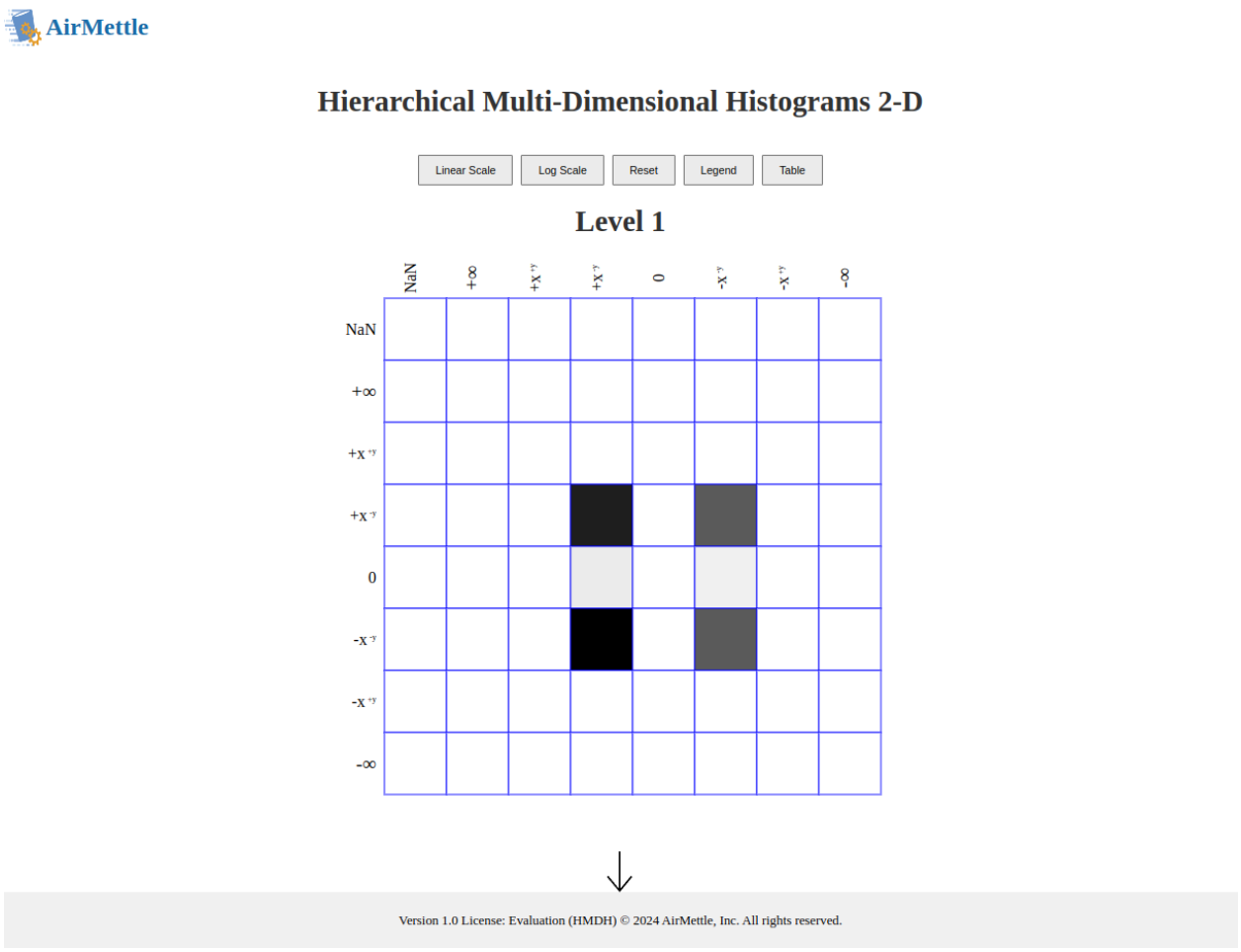


Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Plot for level 3

2D Plots

To generate these plots we used a subset of the [Laghos dataset](). We used the parquet dataset and extracted the columns v_x and v_y which is the velocity in X and Y directions respectively. Each dimension has 100K values and the resulting buffer on disk is ~3KB uncompressed.



Top level plot

	NAN	+Inf	+#, +Exp	+#, -Exp	0	-#, -Exp	-#, +Exp	-Inf
NAN	0	0	0	0	0	0	0	0
+Inf	0	0	0	0	0	0	0	0
+#, +Exp	0	0	0	0	0	0	0	0
+#, -Exp	0	0	0	27397	0	19753	0	0
0	0	0	0	1157	0	635	0	0
-#, -Exp	0	0	0	31295	0	19763	0	0
-#, +Exp	0	0	0	0	0	0	0	0
-Inf	0	0	0	0	0	0	0	0

Top Level Encoding Bits

X: 0 1 1

Y: 0 1 1

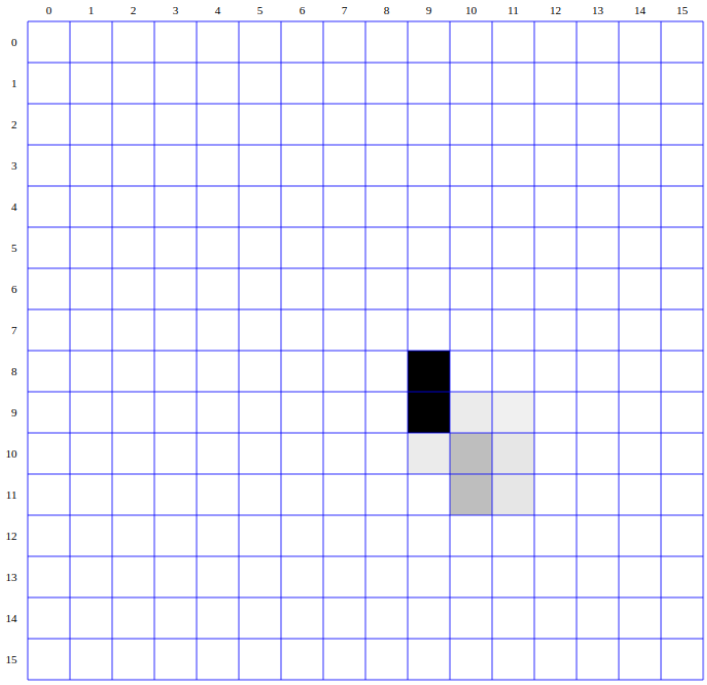
Show Table Drill more →

Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Top level plot (detailed)

Level 2

Show Legend

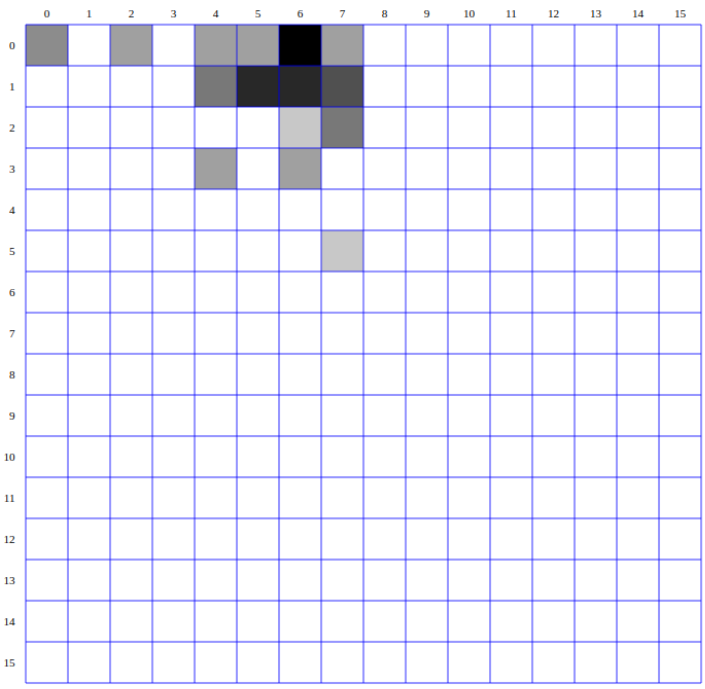


Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Level 2 plot

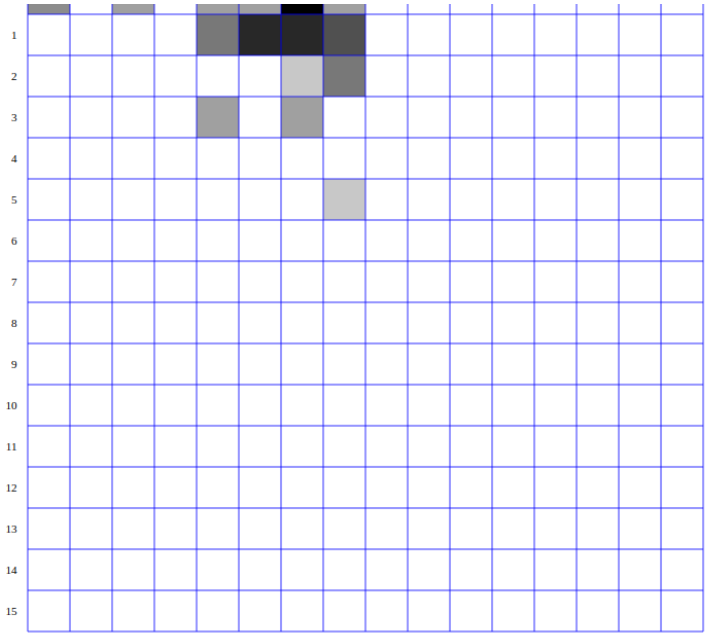
Level 3

Show Legend →



Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Level 3 plot



	M bits				P bits			
X:	1	0	1	1	0	1	1	0
Y:	1	0	0	0	0	0	0	0

Version 1.0 License: Evaluation (HMDH) © 2024 AirMettle, Inc. All rights reserved.

Level 3 plot with M & P bits for each dimension

API

1D API

std::vector<char> generate_1DxF(const FPHArray &array, bool default_mode = true);

- generate_1DxF accepts as parameter a FPHArray type [see Construct FPHArray]
- It returns a compact histogram buffer serialized for efficient disk storage.

2D API

std::vector<char> generate_2DxF(const FPHArray &array1, const FPHArray &array2, bool default_mode = true);

- generate_2DxF accepts as parameters two arrays wrapped in FPHArray type [see Construct FPHArray]
- It returns a compact histogram buffer serialized for efficient disk storage.

Persist generated buffer to disk

Write(buffer_char_vector, "buffer_name");

- Use this to persist the compact serialized data to disk.

Construct FPHArray

- FPHArray buildFPHArray(const double *values, int length);
- FPHArray buildFPHArray(const float *values, int length);
- FPHArray buildFPHArray(const int32_t *values, int length);
- FPHArray buildFPHArray(const int64_t *values, int length);

Use one of the buildFPHArray(...) options to convert your double(float64), float(float32), int64 or int32 values to a FPHArray type. The APIs in 1D API and

2D API will only work with arrays of type FPHArray and not raw c/c++ arrays or vectors.

Commercial Release Preview

The commercial release includes everything from the public release, the APIs listed below and much more:

Generate APIs

Dimension	API	Feature
1	*1DxT*	Small memory and disk usage
	1DxP	Higher precision filing
2	*2DxP*	Higher precision filing
3	*3DxF*	Fast filing
	3DxP	Higher precision filing
	3DxPc	Higher precision with low memory overhead
4	*4DxF*	Fast filing
	4DxP	Higher precision filing
	4DxPc	Higher precision with low memory overhead

Queries APIs

1D query

- maxCount
    - Returns the value with the max count/occurrence in the dataset.

- minCount
    - Returns the value with the min count/occurrence in the dataset.
- percentile
    - Returns the count of values that falls within the input percentile.
- rangeSum
    - Returns the count of values that exist between the start and end points given as input.

UI & E2E Application

The commercial release provides plotting support for all the Generate APIs. We also provide an installable Debian package that provides a command line interface to generate histogram buffers and/or generate the plots directly from the input data file.