# WebDM


shapecomm UG (haftungsbeschränkt)

July 21, 2017

## Contents

# 1 Distribution Matching Specification

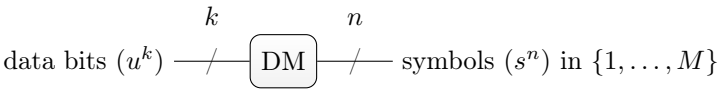


Figure 1: Blackbox model of a distribution matcher.

The specifications of a distribution matcher are summarized in the following.

- **Rate** $= \frac{k}{n} \quad \left[\frac{\text{input bits}}{\text{output symbol}}\right]$
- **Output distribution:** for each $j \in \{1, 2, \ldots, M\}$:
  - Symbol perspective:
$$\Pr(A = j) \approx P_A(j), \quad \text{for each output symbol } A.$$
  - Sequences perspective:
$$\frac{\text{occurrences of } j \text{ in output sequence}}{\text{length of output sequence}} \approx P_A(j).$$
- **Optimized:** $\frac{k}{n} \approx \text{Entropy}(P_A)$.
- **Constant Composition Distribution Matching:** The output sequence is an $n$-type sequence, i.e., it holds $P_A(j) = \frac{x_j}{n}, x_j \in \{0, 1, \ldots, n\}, \sum_{j=1}^{M} x_j = n$.

# 2 Description of the Library

## 2.1 Package contents

The MEX library comes as a zip file with the following files:

```
shapecomm-webdm.zip
├── doc
│   └── doc_webdm.pdf
└── +shapecomm
    ├── +test
    │   └── test_webdm.m
    ├── get_ccdm_params.m
    └── webdm.m
```

## 2.2 System Requirements

The WebDM package requires Matlab 2016a or later. We are currently also developing a version which targets older versions as well. If you are in urgent need of this, please let us know.

## 2.3 Installation

To install the WebDM package, the zip file simply needs to be unpacked. The folder `+shapecomm` is created which contains all necessary Matlab scripts. The WebDM class can then be accessed via

```
>>> dm = shapecomm.webdm(...)
```

To allow access independently from your current directory, it should be added to your MATLAB path and saved for future use:

```
>>> addpath(pwd);
>>> savepath;
```

## 2.4 WebDM object

The library is organized as a MATLAB class which is defined in `webdm.m`. It exports several properties and methods, which will be detailed in the following.

### 2.4.1 Instancing the Class

The constructor of the WebDM class accepts three parameters:

```
>>> dm = shapecomm.webdm(M, n, rate);
```

These have the following meaning:

- `M`: cardinality of the output alphabet, restricted to $\texttt{M} < 32$
- `n`: output length of the DM matcher, restricted to $\texttt{n} < 21600$
- `rate`: desired rate on the output symbols, i.e., bits/output symbol

The restrictions are chosen to ensure a proper functioning as a webservice. A standalone high performance CCDM implementation does not exhibit this restriction. The choice for `M` also imposes an upper bound on the `rate` parameter, which is $\texttt{rate} < \log_2(\texttt{M})$. The resulting distribution on the output symbols will be chosen to resemble a Maxwell-Boltzmann distribution.

### 2.4.2 Properties

After an instance of the WebDM class has been created, the following properties can be accessed:

- `k`: number of input bits
- `n`: number of output symbols
- `M`: size of output alphabet (four amplitudes)
- `pA`: empirical distribution on the output symbols
- `type_seq`: number of occurances of each output symbol
- `host`: server address
- `num_retries`: retries for encode or decode queries (in case of interupted connections)

Except for the `num_retries` property, all other options are **readonly** and should not be modified afterwards, as the server will recognize any wrong configuration and report an error to the client.

### 2.4.3 Encoding

The encoding function exhibits the following signature:

**dm.encode()**

```
s = dm.encode(u)
```

**Signature Description**

- `u`: Vector of length $k$ of zeros and ones that are uniformly distributed and should be encoded.

The encoded symbols `s` will then be taken from the alphabet $\{1, 2, \ldots, \texttt{M}\}$.

### 2.4.4 Decoding

The decoding function exhibits the following signature:

**dm.decode()**

```
uhat = dm.decode(s)
```

**Signature Description**

- `s`: Vector of length $n$ of the output symbols that should be dematched.

# 3 Usage Example

To verify a successful installation and operating of the WebDM package, you can execute a test script:

```
>>> shapecomm.test.test_webdm()
```

It creates a WebDM object for a given configuration and then encodes and decodes as long as it is interrupted by Crtl-C.

# 4 Support

If you encounter any issues or need further support, please contact us at `contact@shapecomm.de`.