
PyEHM

Release 2.0a3.dev2+gb1a3d3f

Lyudmil Vladimirov

May 09, 2023

CONTENTS

1	Installation	3
1.1	Install via pip	3
1.2	Install from source	3
1.3	Development	3
2	API Reference	5
2.1	Core API	5
2.2	Net API	7
2.3	Utils API	11
2.4	Plotting API	11
2.5	Plugins	12
3	Examples	15
3.1	Basic Example	15
3.2	Standard JPDA vs EHM vs EHM2	16
4	License	21
	Bibliography	23
	Index	25

PyEHM is an open-source python package that includes implementations of the Efficient Hypothesis Management (EHM) Algorithms described in [\[EHM1\]](#), [\[EHM2\]](#) and **covered by the patent** [\[EHMPAT\]](#).

INSTALLATION

1.1 Install via pip

You can install the latest release of PyEHM on PyPI by using:

```
python -m pip install pyehm
```

1.2 Install from source

To install the latest version from the GitHub repository:

```
python -m pip install git+https://github.com/sglvladi/pyehm#egg=pyehm
```

1.3 Development

If you are looking to carry out development on PyEHM, you should first clone from GitHub and install with development dependencies by doing the following:

```
git clone "https://github.com/sglvladi/pyehm"  
cd pyehm  
python -m pip install -e .[dev]
```


API REFERENCE

2.1 Core API

The core components of PyEHM are the [EHM](#) and [EHM2](#) classes, that constitute implementations of the EHM [\[EHM1\]](#) and EHM2 [\[EHM2\]](#) algorithms for data association.

The interfaces of these classes are documented below.

class `pyehm.core.EHM`

Efficient Hypothesis Management (EHM)

An implementation of the EHM algorithm, as documented in [\[EHM1\]](#).

static `compute_association_probabilities`(*net*: [EHMNet](#), *likelihood_matrix*: `numpy.ndarray`) → `numpy.ndarray`

Compute the joint association weights, as described in Section 3.3 of [\[EHM1\]](#)

Parameters

- **net** ([EHMNet](#)) – A net object representing the valid joint association hypotheses
- **likelihood_matrix** (`numpy.ndarray`) – A matrix of shape (num_tracks, num_detections + 1) containing the unnormalised likelihoods for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Returns

A matrix of shape (num_tracks, num_detections + 1) containing the normalised association probabilities for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Return type

`numpy.ndarray`

static `construct_net`(*validation_matrix*: `numpy.ndarray`) → [EHMNet](#)

Construct the EHM net as per Section 3.1 of [\[EHM1\]](#)

Parameters

validation_matrix (`numpy.ndarray`) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).

Returns

The constructed net object

Return type

[EHMNet](#)

static run(*validation_matrix*: *numpy.ndarray*, *likelihood_matrix*: *numpy.ndarray*) → *numpy.ndarray*

Run EHM to compute and return association probabilities

Parameters

- **validation_matrix** (*numpy.ndarray*) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).
- **likelihood_matrix** (*numpy.ndarray*) – A matrix of shape (num_tracks, num_detections + 1) containing the unnormalised likelihoods for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Returns

A matrix of shape (num_tracks, num_detections + 1) containing the normalised association probabilities for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Return type

numpy.ndarray

class pyehm.core.**EHM2**

Efficient Hypothesis Management 2 (EHM2)

An implementation of the EHM2 algorithm, as documented in [EHM2].

static compute_association_probabilities(*net*: *EHM2Net*, *likelihood_matrix*: *numpy.ndarray*) → *numpy.ndarray*

Compute the joint association weights, as described in Section 4.2 of [EHM2]

Parameters

- **net** (*EHMNet*) – A net object representing the valid joint association hypotheses
- **likelihood_matrix** (*numpy.ndarray*) – A matrix of shape (num_tracks, num_detections + 1) containing the unnormalised likelihoods for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Returns

A matrix of shape (num_tracks, num_detections + 1) containing the normalised association probabilities for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Return type

numpy.ndarray

static construct_net(*validation_matrix*: *numpy.ndarray*) → *EHM2Net*

Construct the EHM2 net as per Section 4 of [EHM2]

Parameters

validation_matrix (*numpy.ndarray*) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).

Returns

The constructed net object

Return type

EHM2Net

static `construct_tree(validation_matrix: numpy.ndarray) → EHM2Tree`

Construct the EHM2 tree as per section 4.3 of [EHM2]

Parameters

validation_matrix (*numpy.ndarray*) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).

Returns

The constructed tree object

Return type

EHM2Tree

static `run(validation_matrix: numpy.ndarray, likelihood_matrix: numpy.ndarray) → numpy.ndarray`

Run EHM2 to compute and return association probabilities

Parameters

- **validation_matrix** (*numpy.ndarray*) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).
- **likelihood_matrix** (*numpy.ndarray*) – A matrix of shape (num_tracks, num_detections + 1) containing the unnormalised likelihoods for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Returns

A matrix of shape (num_tracks, num_detections + 1) containing the normalised association probabilities for all combinations of tracks and detections. The first column corresponds to the null hypothesis.

Return type

numpy.ndarray

2.2 Net API

The `pyehm.net` module contains classes that implement the structures (nets, nodes, trees) constructed by the *EHM* and *EHM2* classes.

class `pyehm.net.EHMNetNode(layer: int, identity: Set[int])`

A node in the *EHMNet* constructed by *EHM*.

Parameters

- **layer** (*int*) – Index of the network layer in which the node is placed. Since a different layer in the network is built for each track, this also represented the index of the track this node relates to.
- **identity** (*set of int*) – The identity of the node. As per Section 3.1 of [EHM1], “the identity for each node is an indication of how measurement assignments made for tracks already considered affect assignments for tracks remaining to be considered”.

class `pyehm.net.EHM2NetNode(layer: int, track: int, subnet: int, identity: Set[int])`

A node in the *EHM2Net* constructed by *EHM2*.

Parameters

- **layer** (*int*) – Index of the network layer in which the node is placed.

- **track** (`int`) – Index of track this node relates to.
- **subnet** (`int`) – Index of subnet to which the node belongs.
- **identity** (`set of int`) – The identity of the node. As per Section 3.1 of [EHM1], “the identity for each node is an indication of how measurement assignments made for tracks already considered affect assignments for tracks remaining to be considered”.

class `pyehm.net.EHMNet`(*root*: `EHMNetNode`, *validation_matrix*: `numpy.ndarray`)

Represents the nets constructed by *EHM*.

Parameters

- **root** (`EHMNetNode`) – The net root node.
- **validation_matrix** (`numpy.ndarray`) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).

add_edge(*parent*: `EHMNetNode`, *child*: `EHMNetNode`, *detection*: `int`)

Add edge between two nodes, or update an already existing edge by adding the detection to it.

Parameters

- **parent** (`EHMNetNode`) – The parent node, i.e. the source of the edge.
- **child** (`EHMNetNode`) – The child node, i.e. the target of the edge.
- **detection** (`int`) – Index of measurement representing the parent child relationship.

add_node(*node*: `EHMNetNode`, *parent*: `EHMNetNode`, *detection*: `int`)

Add a node to the network.

Parameters

- **node** (`EHMNetNode`) – The node to be added.
- **parent** (`EHMNetNode`) – The parent of the node.
- **detection** (`int`) – Index of measurement representing the parent child relationship.

get_children(*node*: `EHMNetNode`) → `List[EHMNetNode]`

Get the children of a node.

Parameters

- **node** (`EHMNetNode`) – The node whose children should be returned

Returns

List of child nodes

Return type

`list` of `EHMNetNode`

get_edges(*parent*: `EHMNetNode`, *child*: `EHMNetNode`) → `List[int]`

Get edges between two nodes.

Parameters

- **parent** (`EHMNetNode`) – The parent node, i.e. the source of the edge.
- **child** (`EHMNetNode`) – The child node, i.e. the target of the edge.

Returns

Indices of measurements representing the parent child relationship.

Return type`list of int`**get_parents**(*node*: `EHMNetNode`) → `List[EHMNetNode]`

Get the parents of a node.

Parameters**node** (`EHMNetNode`) – The node whose parents should be returned**Returns**

List of parent nodes

Return type`list of EHMNetNode`**property nodes**

The nodes comprising the net

property nodes_forward

The net nodes, ordered by increasing layer

property num_layers

Number of layers in the net

property num_nodes

Number of nodes in the net

property root

The root node of the net

class `pyehm.net.EHM2Net`(*root*: `EHM2NetNode`, *validation_matrix*: `numpy.ndarray`)Represents the nets constructed by *EHM2*.**Parameters**

- **root** (`EHM2NetNode`) – The net root node.
- **validation_matrix** (`numpy.ndarray`) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).

add_edge(*parent*: `EHM2NetNode`, *child*: `EHM2NetNode`, *detection*: `int`)

Add edge between two nodes, or update an already existing edge by adding the detection to it.

Parameters

- **parent** (`EHM2NetNode`) – The parent node, i.e. the source of the edge.
- **child** (`EHM2NetNode`) – The child node, i.e. the target of the edge.
- **detection** (`int`) – Index of measurement representing the parent child relationship.

add_node(*node*: `EHM2NetNode`, *parent*: `EHM2NetNode`, *detection*: `int`)

Add a new node in the network.

Parameters

- **node** (`EHM2NetNode`) – The node to be added.
- **parent** (`EHM2NetNode`) – The parent of the node.
- **detection** (`int`) – Index of measurement representing the parent child relationship.

get_children_per_detection(*node*: [EHM2NetNode](#), *detection*: *int*) → List[[EHM2NetNode](#)]

Get the children of a node for a particular detection.

Parameters

- **node** ([EHM2NetNode](#)) – The node whose children should be returned.
- **detection** (*int*) – The target detection.

get_nodes_per_layer_subnet(*layer*: *int*, *subnet*: *int*) → List[[EHM2NetNode](#)]

Get nodes for a particular layer in a subnet.

Parameters

- **layer** (*int*) – The target layer.
- **subnet** (*int*) – The target subnet.

Returns

List of nodes in the target layer and subnet.

Return type

list of [EHM2NetNode](#)

property nodes

The nodes comprising the net

property nodes_forward

The net nodes, ordered by increasing layer

property nodes_per_track

Dictionary containing the nodes per track

property num_layers

Number of layers in the net

property num_nodes

Number of nodes in the net

property root

The root node of the net

class pyehm.net.[EHM2Tree](#)(*track*: *int*, *children*: List[[EHM2Tree](#)], *detections*: Set[*int*], *subtree*: *int*)

Represents the track tree structure generated by [construct_tree\(\)](#).

The [EHM2Tree](#) object represents both a tree as well as the root node in the tree.

Parameters

- **track** (*int*) – The index of the track represented by the root node of the tree
- **children** (list of [EHM2Tree](#)) – Sub-trees that are children of the current tree
- **detections** (set of *int*) – Set of accumulated detections
- **subtree** (*int*) – Index of subtree the current tree belongs to.

property depth

The depth of the tree

2.3 Utils API

The `pyehm.utils` module contains helper classes and functions.

```
class pyehm.utils.Cluster(tracks: List[int], detections: List[int] = [], validation_matrix: numpy.ndarray =
                           numpy.array([]), likelihood_matrix: numpy.ndarray = numpy.array([]))
```

A cluster of tracks sharing common detections.

Parameters

- **tracks** (list of `int`) – Indices of tracks in cluster
- **detections** (list of `int`) – Indices of detections in cluster. Defaults to an empty list.
- **validation_matrix** (`numpy.ndarray`) – The validation matrix for tracks and detections in the cluster. Defaults to an empty array.
- **likelihood_matrix** (`numpy.ndarray`) – The likelihood matrix for tracks and detections in the cluster. Defaults to an empty array.

```
pyehm.utils.gen_clusters(validation_matrix: numpy.ndarray, likelihood_matrix: numpy.ndarray =
                          numpy.array([])) → List[Cluster]
```

Cluster tracks into groups sharing detections

Parameters

- **validation_matrix** (`numpy.ndarray`) – An indicator matrix of shape (num_tracks, num_detections + 1) indicating the possible (aka. valid) associations between tracks and detections. The first column corresponds to the null hypothesis (hence contains all ones).
- **likelihood_matrix** (`numpy.ndarray`) – A matrix of shape (num_tracks, num_detections + 1) containing the unnormalised likelihoods for all combinations of tracks and detections. The first column corresponds to the null hypothesis. Defaults to an empty array, in which case the likelihood matrices of the generated clusters will also be empty arrays.

```
pyehm.utils.to_nx_graph(obj: EHMNet | EHM2Net | EHM2Tree) → Graph
```

Get a NetworkX representation of a net or tree. Mainly used for plotting.

Parameters

obj (`EHMNet` | `EHM2Net` | `EHM2Tree`) – The object to convert to a NetworkX graph.

Returns

The NetworkX graph representation of the object.

Return type

`networkx.Graph`

2.4 Plotting API

The `pyehm.plot` module contains helper functions for plotting the nets and trees constructed by the `EHM` and `EHM2` classes.

Warning: The plotting functions require `Graphviz` to be installed and on the PATH.

`pyehm.plotting.plot_net(net: EHMNet | EHM2Net, ax: Axes = None, annotate=True)`

Plot the net.

Parameters

- **net** (*EHMNet* | *EHM2Net*) – The net to plot.
- **ax** (`matplotlib.axes.Axes`) – Axes on which to plot the net. If `None`, a new figure and axes will be created.
- **annotate** (`bool`) – Flag that dictates whether to draw node and edge labels on the plotted net. The default is `True`

`pyehm.plotting.plot_tree(tree: EHM2Tree, ax: Axes = None, annotate=True)`

Plot the tree.

Parameters

- **tree** (*EHM2Tree*) – The tree to plot.
- **ax** (`matplotlib.axes.Axes`) – Axes on which to plot the tree. If `None`, a new figure and axes will be created.
- **annotate** (`bool`) – Flag that dictates whether to draw node labels on the plotted tree. The default is `True`

2.5 Plugins

2.5.1 Stone Soup

`class pyehm.plugins.stonesoup.JPDWithEHM(hypothesiser: PDHypothesiser)`

Bases: *JPDA*

Joint Probabilistic Data Association with Efficient Hypothesis Management (EHM)

This is a faster alternative of the standard JPDA algorithm, which makes use of Efficient Hypothesis Management (EHM) to efficiently compute the joint associations. See Maskell et al. (2004) [*EHM1*] for more details.

associate(*tracks, detections, timestamp, **kwargs*)

Associate tracks and detections

Parameters

- **tracks** (set of `stonesoup.types.track.Track`) – Tracks which detections will be associated to.
- **detections** (set of `stonesoup.types.detection.Detection`) – Detections to be associated to tracks.
- **timestamp** (`datetime.datetime`) – Timestamp to be used for missed detections and to predict to.

Returns

Mapping of track to Hypothesis

Return type

mapping of `stonesoup.types.track.Track` : `stonesoup.types.hypothesis.Hypothesis`

class `pyehm.plugins.stonesoup.JPDaWithEHM2`(*hypothesiser*: `PDAHypothesiser`)

Bases: `JPDaWithEHM`

Joint Probabilistic Data Association with Efficient Hypothesis Management 2 (EHM2)

This is an enhanced version of the `JPDaWithEHM` algorithm, that makes use of the Efficient Hypothesis Management 2 (EHM2) algorithm to efficiently compute the joint associations. See Horridge et al. (2006) [EHM2] for more details.

associate(*tracks*, *detections*, *timestamp*, ***kwargs*)

Associate tracks and detections

Parameters

- **tracks** (set of `stonesoup.types.track.Track`) – Tracks which detections will be associated to.
- **detections** (set of `stonesoup.types.detection.Detection`) – Detections to be associated to tracks.
- **timestamp** (`datetime.datetime`) – Timestamp to be used for missed detections and to predict to.

Returns

Mapping of track to Hypothesis

Return type

mapping of `stonesoup.types.track.Track` : `stonesoup.types.hypothesis.Hypothesis`

EXAMPLES

3.1 Basic Example

```
import numpy as np
```

3.1.1 Formulating the possible associations between targets and measurements

Both *EHM* and *EHM2* operate on a `validation_matrix` and a `likelihood_matrix`. The `validation_matrix` is an indicator matrix that represents the possible associations between different targets and measurements, while the `likelihood_matrix` contains the respective likelihoods/probabilities of these associations. Both matrices have a shape (N_T, N_M+1) , where N_T is the number of targets and N_M is the number of measurements.

For example, assume we have the following scenario of 4 targets and 4 measurements (taken from Section 4.4 of [EHM2]):

Target index	Gated measurement indices
0	0, 1
1	0, 1, 2, 3
2	0, 1, 2
3	0, 3, 4

where the null measurement hypothesis is given the index of 0. Then the `validation_matrix` would be a $(4, 5)$ numpy array of the following form:

```
validation_matrix = np.array([[1, 1, 0, 0, 0], # 0 -> 0,1
                             [1, 1, 1, 1, 0], # 1 -> 0,1,2,3
                             [1, 1, 1, 0, 0], # 2 -> 0,1,2
                             [1, 0, 0, 1, 1]]) # 3 -> 0,3,4
```

The `likelihood_matrix` is such that each element `likelihood_matrix[i, j]` contains the respective likelihood of target *i* being associated to measurement *j*. Therefore, based on the above example, the `likelihood_matrix` could be the following:

```
likelihood_matrix = np.array([[0.1, 0.9, 0, 0, 0],
                             [0.1, 0.3, 0.2, 0.4, 0],
                             [0.7, 0.1, 0.2, 0, 0],
                             [0.2, 0, 0, 0.75, 0.05]])
```

3.1.2 Computing joint association probabilities

Based on the above, we can use *EHM* or *EHM2* to compute the joint association probabilities matrix `assoc_matrix` as follows:

```
from pyehm.core import EHM, EHM2

assoc_matrix_ehm = EHM.run(validation_matrix, likelihood_matrix)
print('assoc_matrix_ehm =\n {}'.format(assoc_matrix_ehm))
# or
assoc_matrix_ehm2 = EHM2.run(validation_matrix, likelihood_matrix)
print('assoc_matrix_ehm2 =\n {}'.format(assoc_matrix_ehm2))
```

```
assoc_matrix_ehm =
[[0.17948718 0.82051282 0.          0.          0.          ]
 [0.25925926 0.07692308 0.4045584  0.25925926 0.          ]
 [0.85754986 0.01139601 0.13105413 0.          0.          ]
 [0.35555556 0.          0.          0.55555556 0.08888889]]

assoc_matrix_ehm2 =
[[0.17948718 0.82051282 0.          0.          0.          ]
 [0.25925926 0.07692308 0.4045584  0.25925926 0.          ]
 [0.85754986 0.01139601 0.13105413 0.          0.          ]
 [0.35555556 0.          0.          0.55555556 0.08888889]]
```

Note that both *EHM* and *EHM2* should produce the same results, although *EHM2* should, in principle, be significantly faster for large numbers of targets and measurements.

```
# Check if the probability matrices produced by EHM and EHM2 are equal
print(np.allclose(assoc_matrix_ehm, assoc_matrix_ehm2))
```

```
True
```

Total running time of the script: (0 minutes 0.007 seconds)

3.2 Standard JPDA vs EHM vs EHM2

Both *EHM* and *EHM2* provide an exact solution to the problem posed by the Joint Probabilistic Data Association (JPDA) algorithm. However, even though in the naive implementation JPDA the number of hypotheses, and as a direct consequence the time required to evaluate these, increases exponentially with number of targets and measurements, *EHM* and *EHM2* produce results in sub-exponential time.

3.2.1 Problem formulation

In this example we will be comparing the computational performance of the [EHM](#) and [EHM2](#), against a naive implementation of JPDA, for a relatively dense scenario of 11 targets and 9 measurements. The validation and likelihood matrices for this scenario are defined below (For more information on how these matrices are defined, see the [Basic Example](#)):

```
import itertools
import datetime
import numpy as np

from pyehm.core import EHM, EHM2

validation_matrix = np.array([[1, 1, 1, 0, 1, 0, 1, 1, 0, 0], # 0 -> 0,1,2,4,6,7
                              [1, 1, 0, 1, 1, 1, 1, 1, 0, 0], # 1 -> 0,1,3,4,5,6,7
                              [1, 1, 0, 1, 0, 1, 1, 1, 1, 0], # 2 -> 0,1,3,5,6,7,8
                              [1, 1, 1, 1, 0, 0, 1, 1, 0, 1], # 3 -> 0,1,2,3,6,7,9
                              [1, 0, 1, 1, 0, 0, 0, 0, 1, 0], # 4 -> 0,2,3,8
                              [1, 1, 1, 0, 0, 1, 1, 1, 1, 0], # 5 -> 0,1,2,5,6,7,8
                              [1, 1, 0, 0, 0, 1, 1, 0, 1, 1], # 6 -> 0,1,5,6,8,9
                              [1, 1, 1, 0, 0, 0, 0, 0, 0, 0], # 7 -> 0,1,2
                              [1, 0, 0, 1, 1, 1, 1, 1, 1, 0], # 8 -> 0,3,4,5,6,7,8
                              [1, 0, 0, 0, 1, 0, 1, 1, 1, 0], # 9 -> 0,4,6,7,8
                              [1, 0, 1, 0, 0, 0, 0, 0, 0, 1]]) # 10 -> 0,2,9

likelihood_matrix = np.array([[0.9, 0.13, 0.1, 0, 0.97, 0, 0.94, 0.55, 0, 0],
                              [0.55, 0.31, 0, 0.61, 0.27, 0.38, 0.34, 0.58, 0, 0],
                              [0.61, 0.55, 0, 0.32, 0, 0.25, 0.8, 0.94, 0.62, 0],
                              [0.45, 0.53, 0.61, 0.19, 0, 0, 0.95, 0.61, 0, 0.17],
                              [0.67, 0, 0.79, 0.99, 0, 0, 0, 0, 0.71, 0],
                              [0.51, 0.37, 0.04, 0, 0, 0.53, 0.92, 0.44, 0.95, 0],
                              [0.31, 0.03, 0, 0, 0, 0.08, 0.68, 0, 0.04, 0.31],
                              [0.23, 0.09, 0.21, 0, 0, 0, 0, 0, 0, 0],
                              [0.62, 0, 0, 0.19, 0.17, 0.31, 0.69, 0.89, 0.63, 0],
                              [0.44, 0, 0, 0, 0.53, 0, 0.49, 0.01, 0.31, 0],
                              [0.32, 0, 0.56, 0, 0, 0, 0, 0, 0, 0.23]])
```

3.2.2 EHM vs EHM 2

It is worth noticing that in the above example, targets 7 and 10 are conditionally independent of targets 8 and 9, given target 6. This is because targets 7 and 10 share a common measurement (2), but do not share any measurements with targets 8 or 9, which in turn have common measurements (4, 6, 7, 8). Yet, all of them share measurements with target 6.

This is important since [EHM2](#) takes advantage of this conditional independence to reduce the number of nodes in the constructed net and, as a result, achieve better computational performance than [EHM](#).

To better understand the above, let us examine the number of nodes in the nets produced by the two algorithms:

```
# Net constructed using EHM
net1 = EHM.construct_net(validation_matrix)

# Net constructed using EHM2
```

(continues on next page)

(continued from previous page)

```
net2 = EHM2.construct_net(validation_matrix)

print('No. of nodes in EHM net: {}'.format(net1.num_nodes))
print('No. of nodes in EHM2 net: {}'.format(net2.num_nodes))
```

```
No. of nodes in EHM net: 2050
No. of nodes in EHM2 net: 1317
```

3.2.3 Standard JPDA

Below we define the function `jpda` that computes the joint association probabilities based on the standard JPDA recursion, which performs a full enumeration of all the joint hypotheses.

```
def jpda(validation_matrix, likelihood_matrix):
    num_tracks, num_detections = validation_matrix.shape

    possible_assoc = list()
    for track in range(num_tracks):
        track_possible_assoc = list()
        v_detections = np.flatnonzero(validation_matrix[track, :])
        for detection in v_detections:
            track_possible_assoc.append((track, detection))
        possible_assoc.append(track_possible_assoc)

    # Compute all possible joint hypotheses
    joint_hyps = itertools.product(*possible_assoc)

    # Compute valid joint hypotheses
    valid_joint_hypotheses = (joint_hypothesis for joint_hypothesis in joint_hyps if is_
↪valid_hyp(joint_hypothesis))

    # Compute likelihood for valid joint hypotheses
    valid_joint_hypotheses_lik = dict()
    for joint_hyp in valid_joint_hypotheses:
        lik = 1
        # The likelihood of a joint hypothesis is the product of the likelihoods of its_
↪member hypotheses
        for hyp in joint_hyp:
            track = hyp[0]
            detection = hyp[1]
            lik *= likelihood_matrix[track, detection]
        valid_joint_hypotheses_lik[joint_hyp] = lik

    # Compute the joint association probabilities
    assoc_matrix = np.zeros((num_tracks, num_detections))
    for track in range(num_tracks):
        v_detections = np.flatnonzero(validation_matrix[track, :])
        for detection in v_detections:
            # The joint assoc. probability for a track-detection hypothesis is the sum_
↪of the likelihoods of all
            # joint hypotheses that include this hypothesis
```

(continues on next page)

(continued from previous page)

```

        prob = np.sum([lik for hyp, lik in valid_joint_hypotheses_lik.items() if
→(track, detection) in hyp])
        assoc_matrix[track, detection] = prob
        # Normalise
        assoc_matrix[track, :] /= np.sum(assoc_matrix[track, :])

    return assoc_matrix

def is_valid_hyp(joint_hyp):
    used_detections = set()
    for hyp in joint_hyp:
        detection = hyp[1]
        if not detection:
            pass
        elif detection in used_detections:
            return False
        else:
            used_detections.add(detection)
    return True

```

3.2.4 Comparison

Now we can compare the above against [EHM](#) and [EHM2](#), both in terms of accuracy and computation time. The accuracy comparison is just a safe-guard check to make sure that [EHM](#) and [EHM2](#) produce the same result as the standard JPDA.

```

# EHM
now = datetime.datetime.now()
assoc_matrix_ehm = EHM.run(validation_matrix, likelihood_matrix)
dt_ehm = datetime.datetime.now() - now

# EHM2
now = datetime.datetime.now()
assoc_matrix_ehm2 = EHM2.run(validation_matrix, likelihood_matrix)
dt_ehm2 = datetime.datetime.now() - now

# Standard JPDA
now = datetime.datetime.now()
assoc_matrix_jpda = jpda(validation_matrix, likelihood_matrix)
dt_jpda = datetime.datetime.now() - now

# Check if all results are the same
print(np.allclose(assoc_matrix_jpda, assoc_matrix_ehm, atol=1e-15)
      and np.allclose(assoc_matrix_jpda, assoc_matrix_ehm2, atol=1e-15))

# Compare the execution times
print('JPDA: {} seconds'.format(dt_jpda.total_seconds()))
print('EHM: {} seconds'.format(dt_ehm.total_seconds()))
print('EHM2: {} seconds'.format(dt_ehm2.total_seconds()))

```

```
True
JPDA: 105.682185 seconds
EHM: 0.021402 seconds
EHM2: 0.015098 seconds
```

The above results demonstrate the advantages of using the [EHM](#) and [EHM2](#) classes over the standard JPDA. Both the [EHM](#) and [EHM2](#) algorithms exhibit significant computational gains compared to the standard JPDA, all while producing exactly the same results. We can also observe that [EHM2](#) is noticeably faster than [EHM](#).

Total running time of the script: (1 minutes 45.744 seconds)

LICENSE

PyEHM is licenced under Eclipse Public License 2.0. See [License](#) for more details.

This software is the property of [QinetiQ Limited](#) and any requests for use of the software for commercial use or other use outside of the Eclipse Public Licence should be made to QinetiQ Limited.

The current QinetiQ contact is Richard Lane (rlane1 [at] qinetiq [dot] com).

BIBLIOGRAPHY

- [EHM1] Maskell, S., Briers, M. and Wright, R., 2004, August. Fast mutual exclusion. In Signal and Data Processing of Small Targets 2004 (Vol. 5428, pp. 526-536). International Society for Optics and Photonics
- [EHM2] Horridge, P. and Maskell, S., 2006, July. Real-time tracking of hundreds of targets with efficient exact JPDAF implementation. In 2006 9th International Conference on Information Fusion (pp. 1-8). IEEE
- [EHMPAT] Maskell, S., 2003, July. Signal Processing with Reduced Combinatorial Complexity. Patent Reference:0315349.1

A

add_edge() (*pyehm.net.EHM2Net method*), 9
 add_edge() (*pyehm.net.EHMNet method*), 8
 add_node() (*pyehm.net.EHM2Net method*), 9
 add_node() (*pyehm.net.EHMNet method*), 8
 associate() (*pyehm.plugins.stonesoup.JPDWithEHM method*), 12
 associate() (*pyehm.plugins.stonesoup.JPDWithEHM2 method*), 13

C

Cluster (*class in pyehm.utils*), 11
 compute_association_probabilities() (*pyehm.core.EHM static method*), 5
 compute_association_probabilities() (*pyehm.core.EHM2 static method*), 6
 construct_net() (*pyehm.core.EHM static method*), 5
 construct_net() (*pyehm.core.EHM2 static method*), 6
 construct_tree() (*pyehm.core.EHM2 static method*), 6

D

depth (*pyehm.net.EHM2Tree property*), 10

E

EHM (*class in pyehm.core*), 5
 EHM2 (*class in pyehm.core*), 6
 EHM2Net (*class in pyehm.net*), 9
 EHM2NetNode (*class in pyehm.net*), 7
 EHM2Tree (*class in pyehm.net*), 10
 EHMNet (*class in pyehm.net*), 8
 EHMNetNode (*class in pyehm.net*), 7

G

gen_clusters() (*in module pyehm.utils*), 11
 get_children() (*pyehm.net.EHMNet method*), 8
 get_children_per_detection() (*pyehm.net.EHM2Net method*), 9
 get_edges() (*pyehm.net.EHMNet method*), 8
 get_nodes_per_layer_subnet() (*pyehm.net.EHM2Net method*), 10

get_parents() (*pyehm.net.EHMNet method*), 9

J

JPDWithEHM (*class in pyehm.plugins.stonesoup*), 12
 JPDWithEHM2 (*class in pyehm.plugins.stonesoup*), 12

N

nodes (*pyehm.net.EHM2Net property*), 10
 nodes (*pyehm.net.EHMNet property*), 9
 nodes_forward (*pyehm.net.EHM2Net property*), 10
 nodes_forward (*pyehm.net.EHMNet property*), 9
 nodes_per_track (*pyehm.net.EHM2Net property*), 10
 num_layers (*pyehm.net.EHM2Net property*), 10
 num_layers (*pyehm.net.EHMNet property*), 9
 num_nodes (*pyehm.net.EHM2Net property*), 10
 num_nodes (*pyehm.net.EHMNet property*), 9

P

plot_net() (*in module pyehm.plotting*), 11
 plot_tree() (*in module pyehm.plotting*), 12

R

root (*pyehm.net.EHM2Net property*), 10
 root (*pyehm.net.EHMNet property*), 9
 run() (*pyehm.core.EHM static method*), 5
 run() (*pyehm.core.EHM2 static method*), 7

T

to_nx_graph() (*in module pyehm.utils*), 11