User Guide

d.k.razsolkov

February 2021

1 Installing

This project uses conda, so make sure to have it installed first.

- Create a local folder for the Project
- Clone the Project repository to your machine

```
$ git clone https://git.rwth-aachen.de/chan.yong.lee/conformance-checking.git
```

- Open the Code folder of the Project

```
$ cd conformance-checking/Code
```

- Create and activate the conda environment

```
$ conda env create -f conda-env.yml
$ conda activate conformance-checking
```

- Run the tox command to install all dependencies and run the test-suite

```
$ tox
```

2 API

This project comes with an API Documentation, which can be found here. The API Documentation gives information on the different functionalities of the project, function names, parameters and return types.

3 Examples

To see some examples of how the project functions a set of example scripts have been created in the `Code/examples` directory. To execute them you need to be in the `Code` directory and inside the activated conda environment. The examples can be run with the following commands:

- Custom Algorithm: In this example, we take the length of a trace as its embedding. The dissimilarity is then the relative difference in the length of a trace. This is a very basic implementation for demonstration purposes.

```
$ python -m examples.custom_algorithm
```

- Imports: This is an example of importing an event log from a .xes file as well as import a petri net and generate its playout.

```
$ python -m examples.imports
```

- WMD and ICT: Calculates the WMD and ICT between two traces.

```
$ python -m examples.wmd_ict
```

- Embeddings: Examplifies the results from the calculation of activity and trace embeddings.

```
$ python -m examples.embeddings
```

- Algorithms: This example shows the results of the implemented algorithms *Act2VecWmdConformance*, *Act2VecIctConformance* and *Trace2VecCosineConformance*.

```
$ python -m examples.algorithms
```

4 Known Issues

At the time of release a known issue with this project is the lack of support for M1 Macs. This is due to the incompatibility of the tensorflow version used with the M1 Macs and is up to Apple or Tensorflow to resolve in the future.

5 Implementing a custom algorithm

To create a custom algorithm you must follow the structure of the *algorithms.py* file, located in `Code/conformance-checking/`.

Your algorithm should be declared as a class that inherits from the *EmbeddingConformance* class. The logic of each implementation is defined in _calc_embeddings() and _calc_dissimilarity (). For further information and structure reference the API Documentation.