MSc Computer Science

COMP702 Design & Specification

Open-Source Temporal Networks Library

Seán O'Callaghan

UNIVERSITY OF LIVERPOOL

Overview

Introduction

Specification

Design

Timeline

Requirements

UNIVERSITY OF LIVERPOOL

Introduction

Why?

Open-source Temporal Networks Library written in Python.

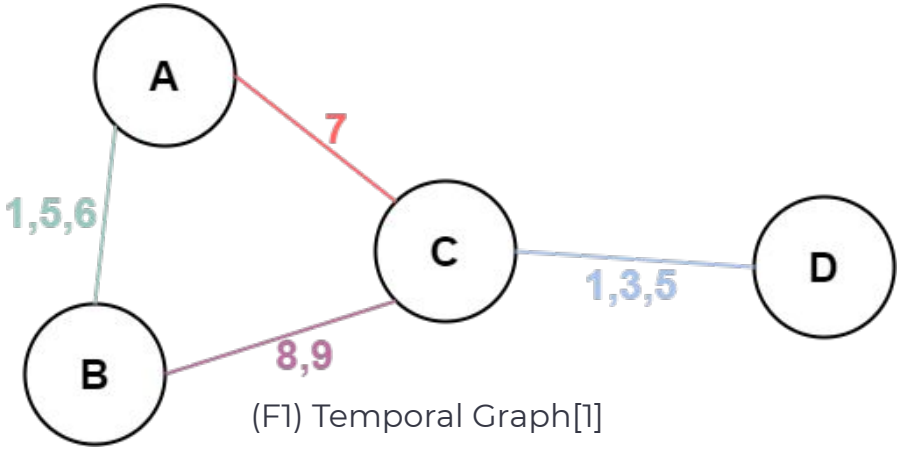
Temporal Network

A temporal network is defined as a network whose links change over time.

Many real-world systems can be modelled as temporal networks [2,3].

The modelling and analysis of such networks can provide useful information.

A purposeful, convenient library to enable this analysis would be advantageous.



(F1) Temporal Graph[1]

Open-source

Enables collaboration

Potential for rapid development

Promotes knowledge share

NetworkX [4]

(F2) NetworkX logo[4]

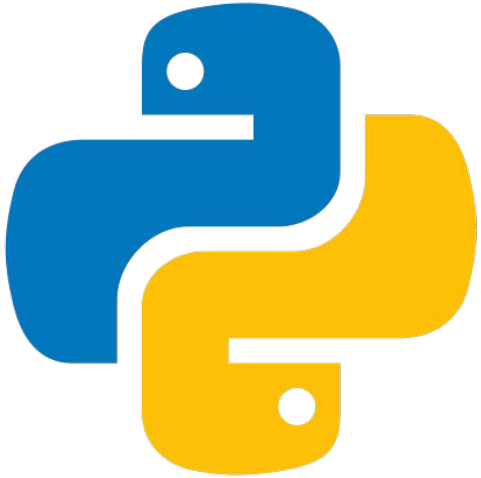
Python

High-level, powerful

Easy to learn

Productive

Popular



(F3) Python logo[5]

UNIVERSITY OF LIVERPOOL

Specification

What?

Aim

Core functionality

Extendable functionality

Language & platform

Open-source

(F4) Package specs[6]

Core functionality

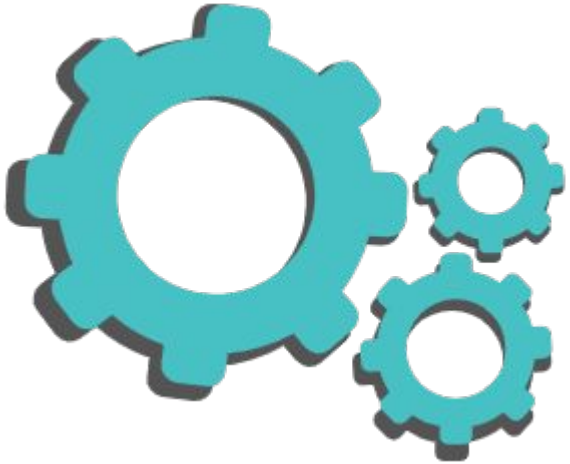
Input handling

Graph creation & generation

Graph functions

Graph visualization

Graph analysis



(F5) Core functionality[6]

UNIVERSITY OF LIVERPOOL

Design

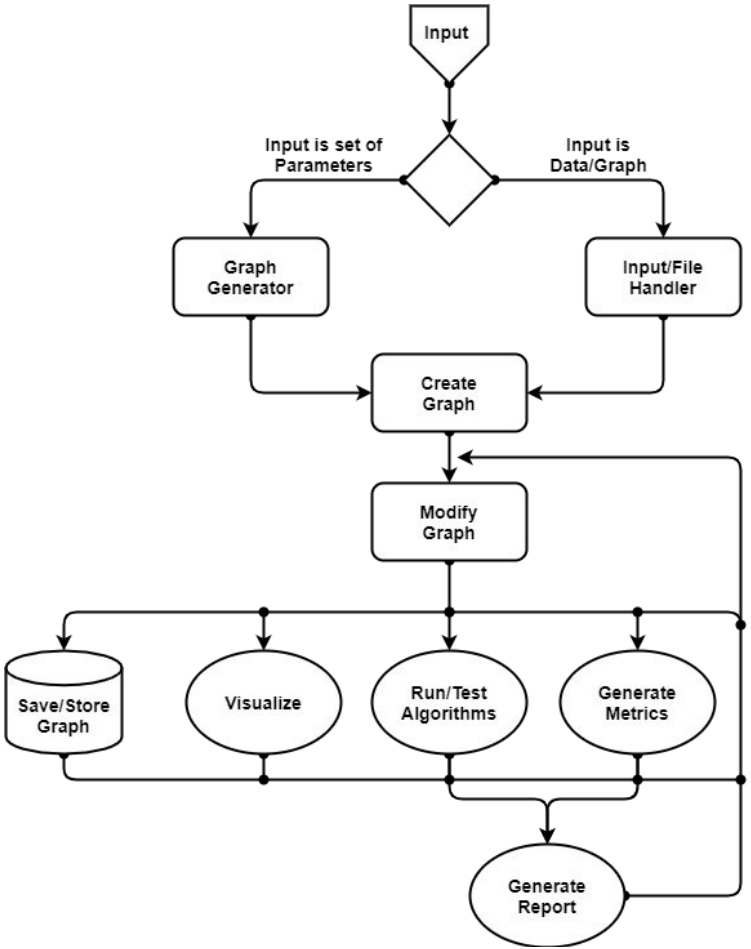
Design Flowchart

Input Handling

Graph Structure

Graph analysis & storage

Graph metrics



(F6) Flowchart[1]

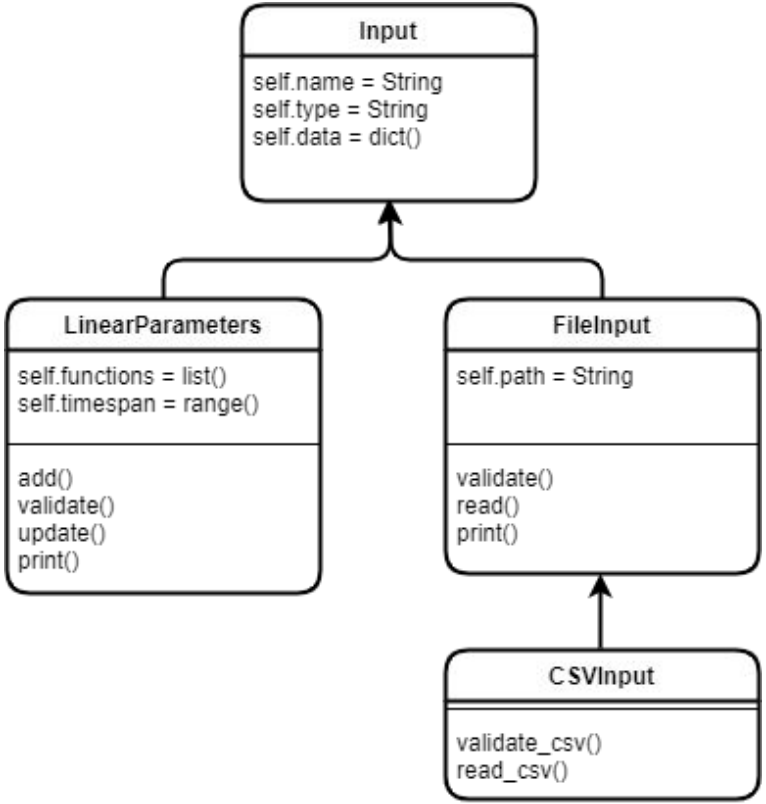
Input Handling

```python
class CSVInput(Input):
    """
        A class which handles CSV-based temporal network/data inputs.
    """

    def __init__(self, name, atype, path):
        super().__init__(name, atype)
        self.path = path
        self.data = {}
        self.read()


    def read(self):
        with open(self.path, newline='') as csvfile:
            reader = csv.DictReader(csvfile)
            n = 0
            data = {}
            for row in reader:
                data[n] = {}
                data[n]['source'] = row['source']
                data[n]['sink'] = row['sink']
                data[n]['time'] = row['time']
                n += 1
        self.data = data
```

```
source,sink,time
a,e,2
d,b,9
a,c,10
b,c,9
d,c,7
b,d,6
e,b,4
a,d,7
d,c,8
b,c,10
```

network.csv

(F7) CSVInput[1]



(F8) Input UML[1]

Graph Structure

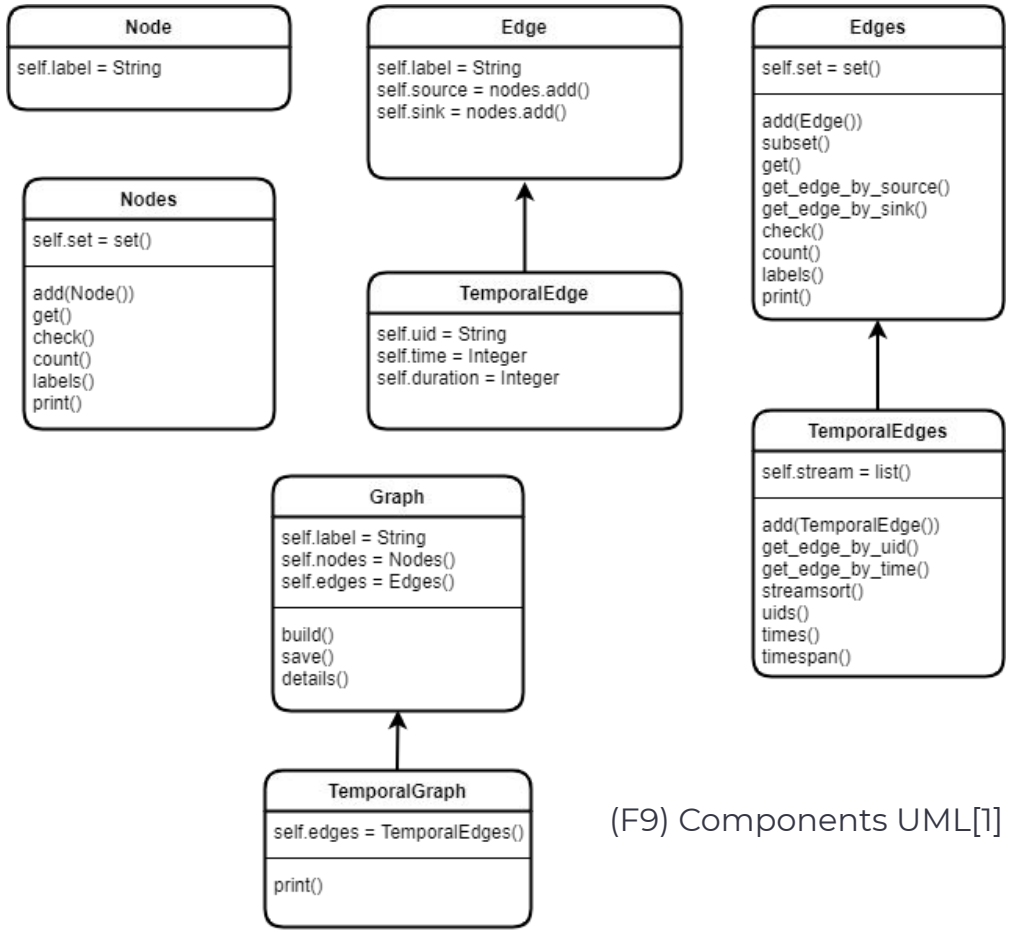
Node

Edge, TemporalEdge

Nodes

Edges, TemporalEdges

Graph, TemporalGraph



(F9) Components UML[1]

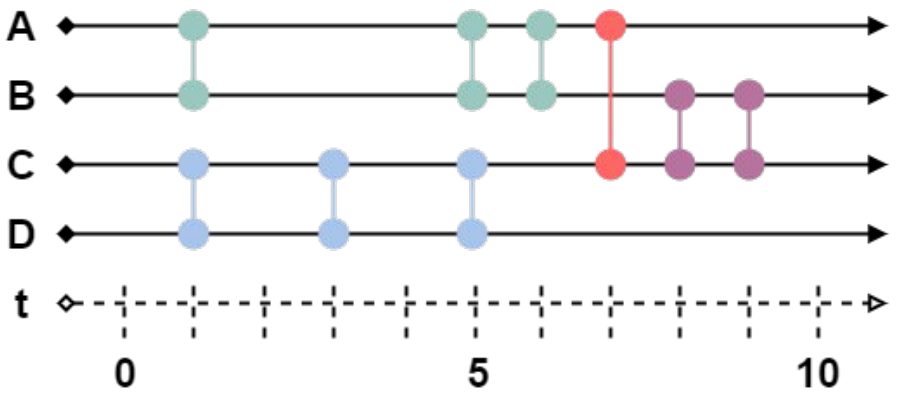
Visualization

```
>>> graph.print()

    5 nodes;
      c a b e d


       ad db bc ac eb bd ae dc
  2 |  -  -  -  -  -  -  +  -
  4 |  -  -  -  -  +  -  -  -
  6 |  -  -  -  -  -  +  -  -
  7 |  +  -  -  -  -  -  -  +
  8 |  -  -  -  -  -  -  -  +
  9 |  -  +  +  -  -  -  -  -
 10 |  -  -  +  +  -  -  -  -

>>>
```

(F10) Edges-matrix[1]

(F11) Edges print[1]

```
def print(self, start=None, end=None):
    print("\n{:5} {}".format(" ", " ".join(self.labels())) )
    for i in self.timespan(start, end):
        active = self.get_edge_by_time(i).labels()
        if not active:
            continue
        row = ['-']*len(self.labels())
        for label in active:
            index = self.labels().index(label)
            row[index] = '+'
        print("{:3} | {:2}".format(i, "  ".join(map(str, row))) )
    print()
```



(F12) Splice plot[1]

Analysis

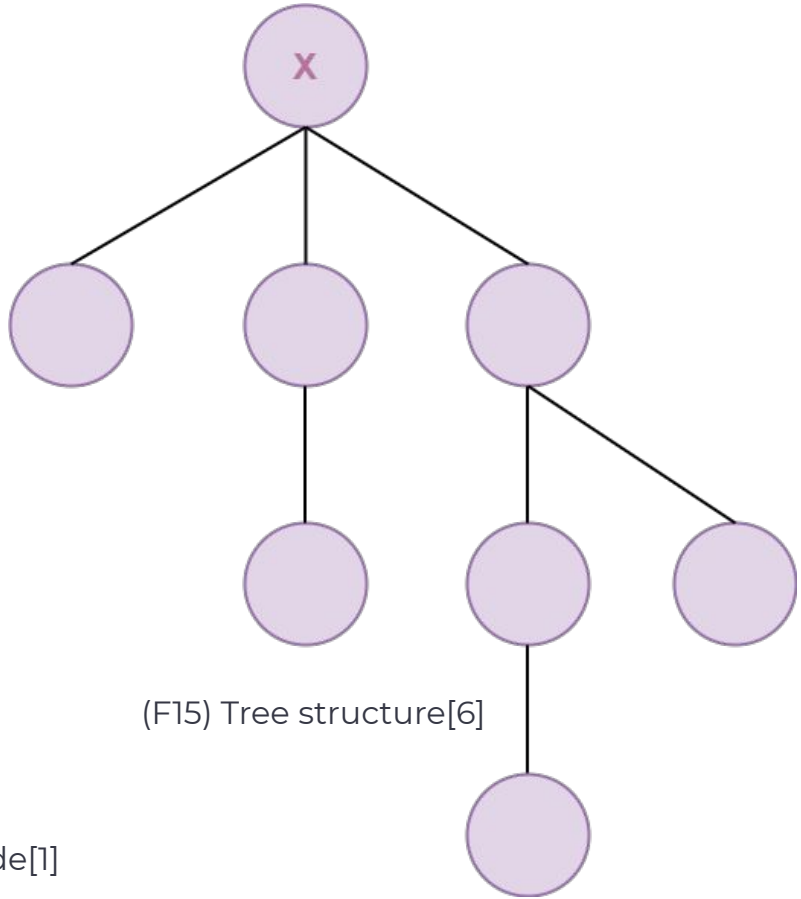
```
# calculate foremost time (to be added)
foremost = {}
a = graph.nodes.get('a')
start = graph.edges.firsttime()
end = graph.edges.lifetime()

for node in graph.nodes.set:
    foremost[node.label] = {}
    foremost[node.label]['time'] = float('inf')
    foremost[node.label]['source'] = ''

foremost[a.label]['time'] = start
foremost[a.label]['source'] = a.label

for edge in graph.edges.stream:
    if edge.time + edge.duration and edge.time >= foremost[edge.source.label]['time']:
        if edge.time + edge.duration < foremost[edge.sink.label]['time']:
            foremost[edge.sink.label]['time'] = edge.time + edge.duration
            foremost[edge.sink.label]['source'] = edge.source.label
    elif edge.time >= end:
        break
```

(F13) Foremost tree implementation[1]

Algorithm: Foremost time/foremost path algorithm (s4.2 [8], modified).

Input: An edge-stream representation of the graph, a time interval (t_α, t_ω), a root node x.
Output: A foremost tree for root node x.
initialize $foremostTree[v][time] = \infty$ for all $v \in V$,
$foremostTree[x][time] = t_\alpha$, and $foremostTree[x][source] = x$;
foreach *edge* $e = (u, v, t, \lambda)$ *in the edge stream* **do**
  if $t + \lambda \leq t_\omega$ *and* $t \geq foremostTree[u][time]$ **then**
    if $t + \lambda < foremostTree[v][time]$ **then**
      $foremostTree[v][time] \longleftarrow t + \lambda$;
      $foremostTree[v][source] \longleftarrow u$;
  else if $t \geq t_\omega$ **then**
    break;
end
return *foremostTree for root node* x;

(F14) Foremost tree pseudo-code[1]



(F15) Tree structure[6]

Evaluation

Standardized testing format

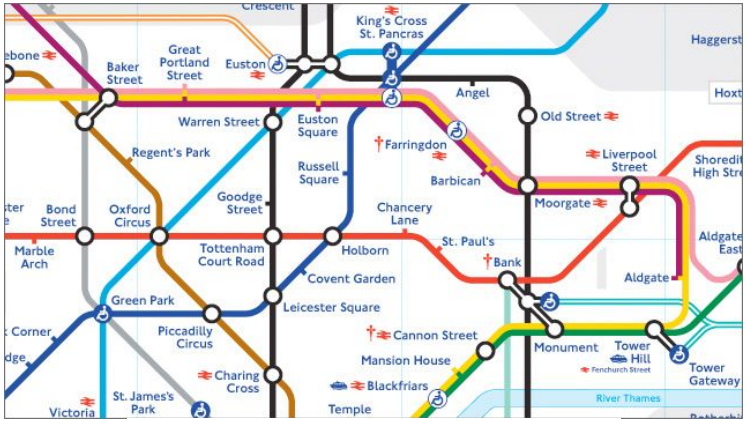
Suitable temporal networks test case(s)

A simple unit test

```python
# test_port1.py

import unittest
from portfolio1 import Portfolio

class PortfolioTest(unittest.TestCase):
    def test_buy_one_stock(self):
        p = Portfolio()
        p.buy("IBM", 100, 176.48)
        assert p.cost() == 17648.0
```

```
$ python -m unittest test_port1
.
----------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

(F16) Unit test[7]



(F17) Transport Network[8]

Open-source checklist

Code

Documentation

License

Library name

(F18) Open-source[9]

Notes

Initial library

Continuous time

Undirected graphs

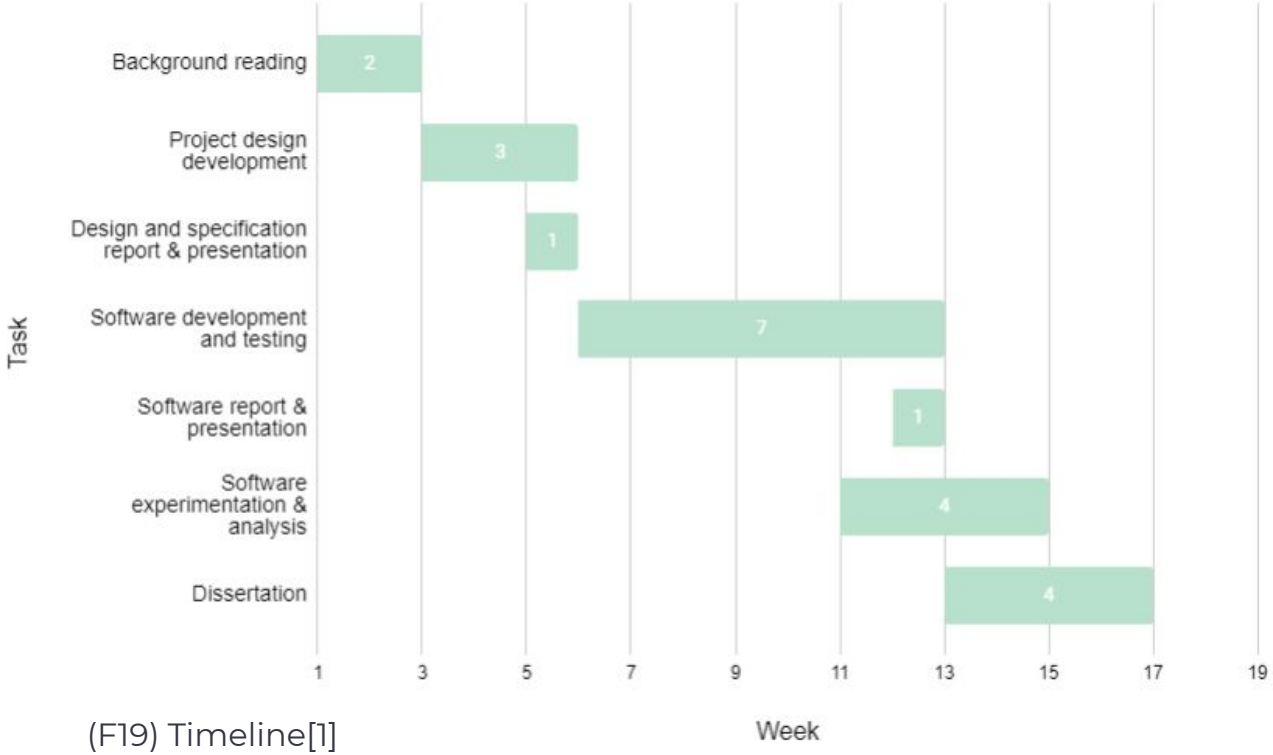
Modular architecture

Efficiency

UNIVERSITY OF LIVERPOOL

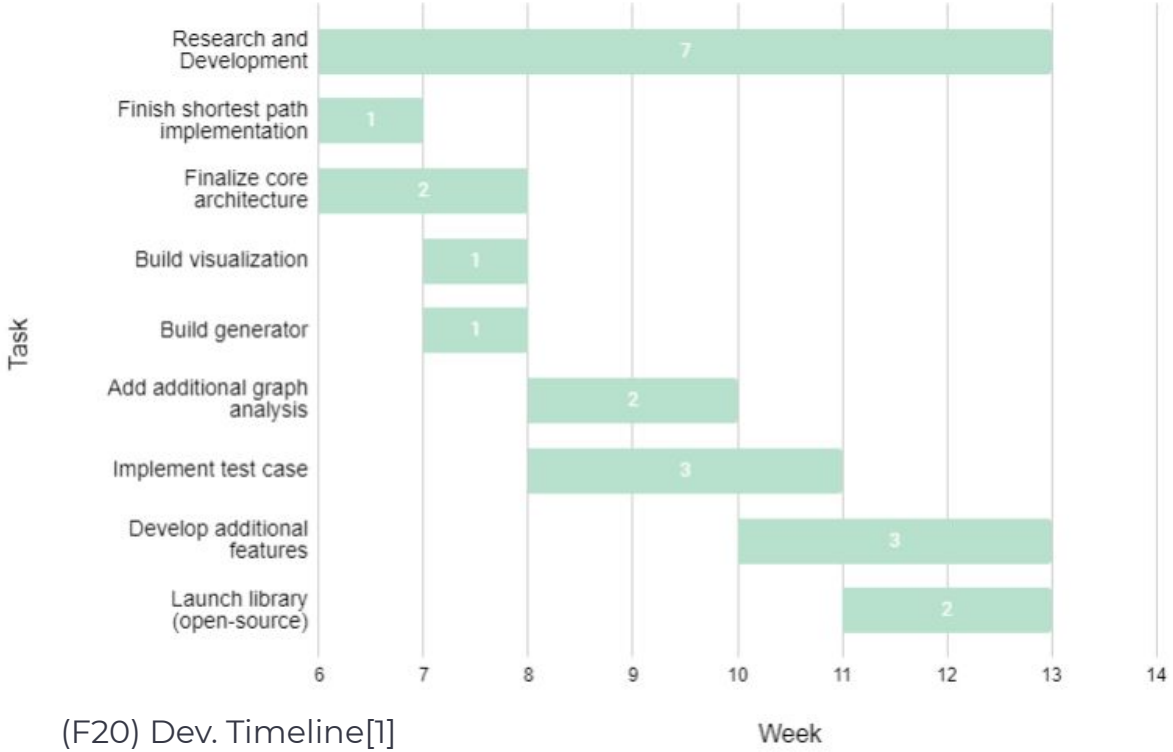
Timeline

When?



(F19) Timeline[1]

Software development and testing



(F20) Dev. Timeline[1]

UNIVERSITY OF LIVERPOOL

Requirements

With?

Python 3.7[5]

- Numpy[12]
- Matplotlib[13]

Git[11]

Github[10]

Open-source[9]

Thanks!

[1]

‘soca-git/COMP702-Temporal-Networks-Library’, *GitHub*. https://github.com/soca-git/COMP702-Temporal-Networks-Library (accessed Jul. 02, 2020).

[2]

J. Tang, M. Musolesi, C. Mascolo, and V. Latora, ‘Temporal distance metrics for social network analysis’, in *Proceedings of the 2nd ACM workshop on Online social networks*, Barcelona, Spain, Aug. 2009, pp. 31–36, doi: 10.1145/1592665.1592674.

[3]

S. Lee, L. E. C. Rocha, F. Liljeros, and P. Holme, ‘Exploiting Temporal Network Structures of Human Interaction to Effectively Immunize Populations’, *PLoS ONE*, vol. 7, no. 5, p. e36439, May 2012, doi: 10.1371/journal.pone.0036439.

[4]

‘NetworkX — NetworkX documentation’. https://networkx.github.io/ (accessed Jun. 30, 2020).

[5]

‘Python Release Python 3.7.7’, *Python.org*. https://www.python.org/downloads/release/python-377/ (accessed Jul. 03, 2020).

[6]

‘All your designs’, *Canva*. https://www.canva.com/folder/all-designs (accessed Jul. 02, 2020).

[7]

‘Ned Batchelder: Getting Started Testing’. https://nedbatchelder.com/text/test0.html (accessed Jul. 02, 2020).

[8]

London *et al.*, ‘Free London travel maps’, *visitlondon.com*. https://www.visitlondon.com/traveller-information/getting-around-london/london-maps-and-guides/free-london-travel-maps (accessed Jul. 02, 2020).

[9]

‘Starting an Open Source Project’, *Open Source Guides*. https://opensource.guide/starting-a-project/ (accessed Jun. 30, 2020).

[10]

‘GitHub; Build software better, together’, *GitHub*. https://github.com (accessed Jul. 02, 2020).

[11]

‘Git’. https://git-scm.com/ (accessed Jul. 02, 2020).

[12]

‘NumPy’. https://numpy.org/ (accessed Jul. 02, 2020).

[13]

‘Matplotlib: Python plotting — Matplotlib 3.2.2 documentation’. https://matplotlib.org/index.html (accessed Jul. 02, 2020).