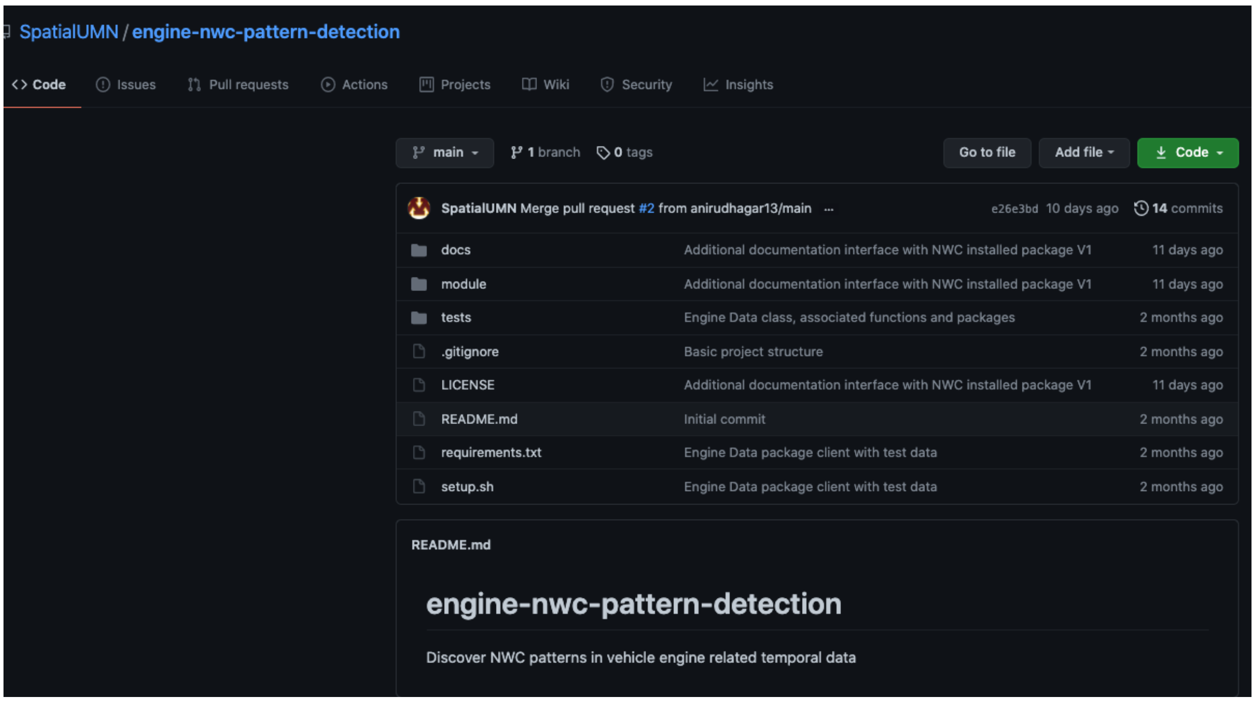
Table of Contents

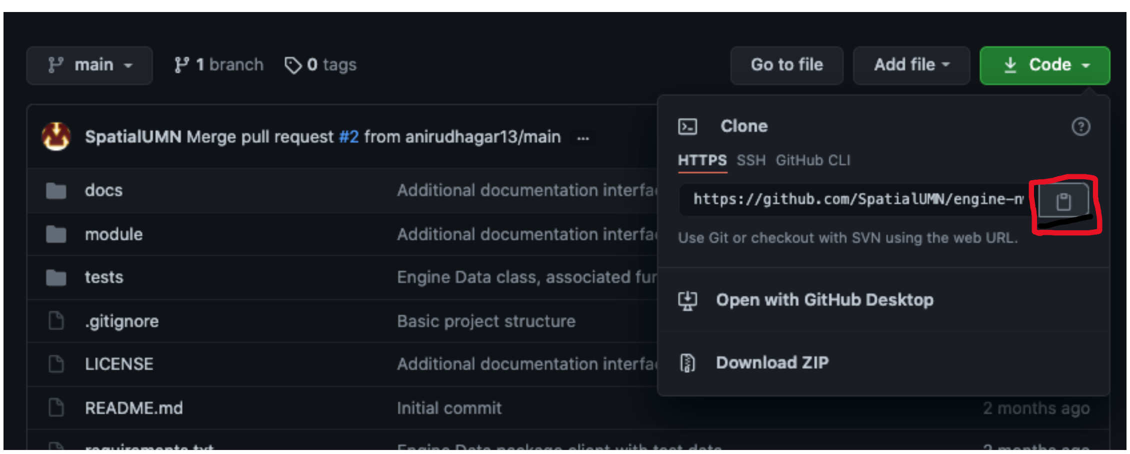
Co-occurrence Pattern Detection

Package Setup

- Go to the research projects page on the UMN Spatial Group Research website.
- Click on `*Non-compliant Cooccurrence Pattern Mining in Temporal Data*` project.
- Then click on GitHub Link under the `*engine_nwc_pattern_detection*` subheading.
- This would take you to the Github page of the Engine NWC data package.



- Click on green code button, then click on clipboard icon to copy the git url.



- Open a terminal and type: `git clone` https://github.com/SpatialUMN/engine-nwc-pattern-detection.git
- Now that the repository is cloned, type: `cd engine-nwc-pattern-detection/`
- Once inside the folder, verify that you have Python 3.7.4 installed, by typing: `python3 -version`

```
Anirudhs-MacBook-Pro:engine-nwc-pattern-detection anirudhagarwal$ python3 --version
Python 3.7.4
```

- To setup the package, simply type: `sh setup.sh`

```
  Using cached tqdm-4.60.0-py2.py3-none-any.whl (75 kB)
Requirement already satisfied: pandas==1.2.4 in ./NWC_env/lib/python3.7/site-packages (from nwc-pattern-miner) (1.2.4)
Requirement already satisfied: numpy>=1.16.5 in ./NWC_env/lib/python3.7/site-packages (from pandas==1.2.4->nwc-pattern-miner) (1.20.2)
Requirement already satisfied: python-dateutil>=2.7.3 in ./NWC_env/lib/python3.7/site-packages (from pandas==1.2.4->nwc-pattern-miner) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in ./NWC_env/lib/python3.7/site-packages (from pandas==1.2.4->nwc-pattern-miner) (2021.1)
Requirement already satisfied: six>=1.5 in ./NWC_env/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas==1.2.4->nwc-pattern-miner) (1.16.0)
Installing collected packages: tqdm, nwc-pattern-miner
Successfully installed nwc-pattern-miner-0.1 tqdm-4.60.0
Anirudhs-MacBook-Pro:engine-nwc-pattern-detection anirudhagarwal$
```

- The above would install all dependencies in a virtual environment `*NWC_env*`.
- Now to activate virtual environment, run command: `source ./NWC_env/bin/activate`
- This would activate your environment, and you can see a small symbol before in the terminal for that.

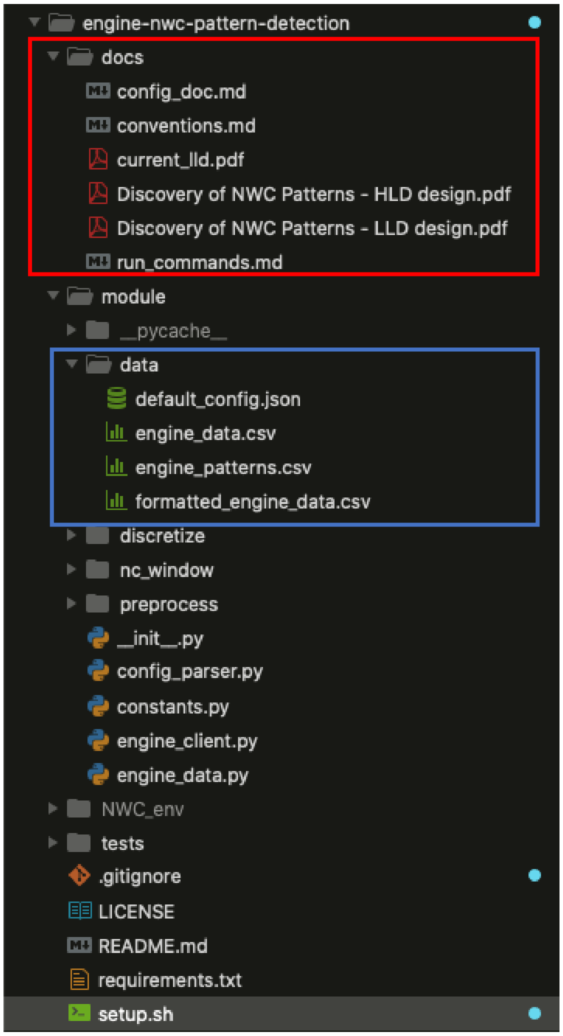
```
Anirudhs-MacBook-Pro:engine-nwc-pattern-detection anirudhagarwal$ source ./NWC_env/bin/activate
(NWC_env) Anirudhs-MacBook-Pro:engine-nwc-pattern-detection anirudhagarwal$
```

- Now you can simply run the package to mine patterns using command: `python -m module.engine_client`
- The package would start running as follows:

```
******************** | Engine Data Preperation | ********************

ount of Invalid Seq Indexes:  4521

-------------------- | Completed Engine Data Preprocessing | --------------------


-------------------- | Completed Finding Anomalous Windows:  46485 | --------------------

          engrpm       EGRkgph        MSPhum         EngTq
ount  55255.000000  55255.000000  55255.000000  55255.000000
ean    1231.610074     70.490710     54.885839    653.006809
td      185.198750     41.297286     12.368099    409.676105
in      800.058000      0.000000     38.005000      0.000000
5%     1089.235000     45.508350     44.234300    288.900000
0%     1229.250000     66.523700     52.285300    593.850000
5%     1357.170000     87.650000     62.601400    976.046000
ax     2084.930000    395.688000     90.000000   1540.800000

-------------------- | Completed Discretizing Feature Columns | --------------------


******************** | Engine Data Formatted and Saved | ********************


******************** | Counting pattern occurences | ********************


-------------------- | Completed Sequence Hashing for Support Count (HashSize): 0.000248 MB | --------------------


******************** | Processing Anomalous Windows | ********************

24%|                                                   | 11378/46485 [00:18<00:52, 668.56it/s]
-------------------- | Number of patterns enumerated: 10000 | Memory size: 0.746152 MB | --------------------

00%|                                                   | 46485/46485 [01:02<00:00, 749.55it/s]

-------------------- | Completed Mining, Pattern Enumerations saved: (682248 / 698970) | --------------------


******************** | Formatting Enumerated Patterns (16722) as Output via: (topk) | ********************


******************** | Engine Patterns Mined and Saved | ********************

NWC_env) Anirudhs-MacBook-Pro:engine-nwc-pattern-detection anirudhagarwal$
```

Data / Resources within Repositories

• To control the experiments, the package has a `*default_config.json*`, which can be edited to alter the experiments.



- (Red): Contains all the documentation related to the package.
  - *config_doc:* Details of each parameter in the config file
  - *run_commands:* Already covered in the above section, as to how to run the package.

- (Blue): Contains all the data files in the package.
  - *default_config.json:* Used to modity the experiements (make sure all columns used in this are present in the corresponding input data file).
  - *engine_data.csv:* Input file to the package
  - *formatted_engine_data.csv:* Discrete data generated by specific engine package
  - *engine_patterns.csv:* TopK patterns mined by the package

Terminology

- **Event:** Discretized / continuous explainable variables (dimensions)
- **Event Pattern:** Sequence of events (for one or multiple dimensions) in a time series
- **NWC:** Non-compliant window cooccurrence i.e., a zonal function over a target defying expectation.
- **NWC pattern:** All candidate patterns (event sequences) from either one or many explainable variables, that occur with or in delta time interval of non-compliant window.

- **JoinSet Cardinality:** No of times a pattern |C| co-occurs in delta time interval of an anomalous window |W| i.e., $|C \bowtie W_n|$.

- **Local Upper Bound (UBmax):** $\frac{Num\ of\ readings\ |T|}{Anom.\ Windows\ |W_n|} * \frac{Max.\ Subset/Leaf\ Joinset\ Count\ (Upper_{loc}\ |C \bowtie W_n|)}{Superset\ Pattern\ Count\ (Lower\ |C|)}$

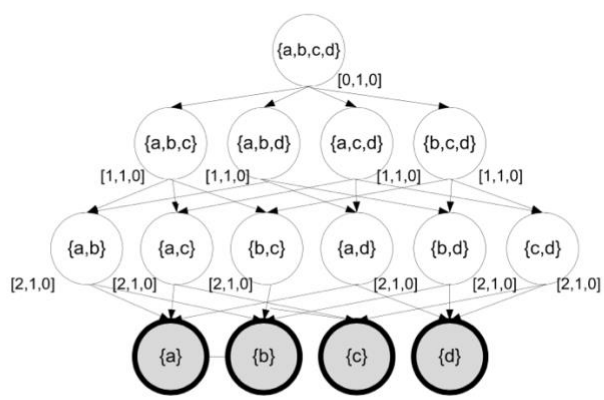
- **Lattice Upper Bound (UBmin):** $\frac{Num\ of\ readings\ |T|}{Anom.\ Windows\ |W_n|} * \frac{Min.\ Subset/Leaf\ Joinset\ Count\ (Upper_{loc}\ |C \bowtie W_n|)}{Superset\ Pattern\ Count\ (Lower\ |C|)}$

- **Support:** Denotes popularity of the pattern in data i.e., $\frac{Joinset\ Card.\ |C \bowtie W_n|}{Num\ of\ readings\ |T|}$
- **Confidence:** $\frac{Unique\ Join\ set\ Card.\ |C \bowtie_0 W_n|}{Pattern\ Count\ |C|}$ i.e., No. of times a pattern co-occurs with an anomalous window (not taking delta into account).

- **Ripley's-k:** $\frac{Num\ of\ readings\ |T|}{Anom.\ Windows\ |W_n|} * \frac{Joinset\ Card\ |C \bowtie W_n|}{Pattern\ Count\ |C|}$

- **supersetCount:** The maximum cardinality found so far (including node itself) of a superset pattern of this node.

- **Lattice graph:** Representing all combinations of dimensions in a hierarchical fashion to analyze all possible combination of candidate patterns. E.g., for 4 dimensions, lattice graph would look like:



MTNMiner: A Multi-Parent Tracking Approach for Mining NWC patterns

- A simplified and earlier version of BDNMiner algorithm.
- Has the same control flow, but no bottom-up pruning.
- The only difference with Top-down pruning in BDNMiner are:
  - Used queue to perform a BFS traversal (adding a child when it's last parent is being visited).
  - Maintaining a visited parent count at each child (to avoid repetition).

BDNMiner: A Bi-Directional approach for mining NWC patterns

- Non-compliant Window Co-occurrence (NWC) pattern detection in time series data.
- The algorithm tries to find candidate patterns that co-occur with anomalous behavior of a target feature in time series data.
- The main contribution is pruning of the combination tree (called **lattice**) for each comparative analysis made with an anomalous window.
- Top-down pruning based on **Upper Bound** and bottom-up pruning based on **Support i.e.** (**Apriori algorithm** from association analysis).

pattern_mining:

```
for each pattern length in input range:
      for each anomalous window:
            for each lag value [0, lag]:

                  create/clone lattice_graph

                  # To get leavesjoint count for upper bound calculation
                  leaf_join_set_counts <- leaf_enumeration()

                  if all leaves pruned via min_support:
                        continue

                  # To get superset count for nodes at level n-1
                  root_enumeration()

                  while !one_level_pruned & top > bottom:
                        for each node at top level:
                              enumerate_with_upper_bound_pruning()

                        top = top – 1

                        if top still > bottom:
                              for each node at bottom level:
                                    # Check for one_level_pruned here
                                    enumerate_with_min_support()

                              bottom = bottom + 1
```

enumerate_with_upper_bound_pruning:

```
If node pruned by UB:
      return

If node pruned by support: - Step does not make sense
      for each non-leaf child not pruned by UB:
            propagate supersetcount to children
      return

UB_lattice <- tight_upper_bound(max_bottom_level_joinsetcount, supersetcount)
If UB_lattice < ε:
      prune_all_children()
      return

UB_local <- tight_upper_bound(min_bottom_level_joinsetcount, supersetcount)
elif UB_local < ε:
      # Are able to save pattern expansion of node in this way

      for each non-leaf child not pruned by UB:
            propagate supersetcount to children
      return

else:
      # No pruning occurs
      expand pattern from the index, using dimensions at the node

      If pattern not enumerated:
            count_the_pattern()

            If pattern_support > min_support & cross_k > ε:
                  Output_the_pattern

            for each non-leaf child not pruned by UB:
                  propagate supersetcount to children

      else:
            # As previously enumerated
            prune_all_children()
```

enumerate_with_min_support:

```
If node pruned by support:
      return True

If node pruned by UB:
      return False

If pattern not enumerated:
      count_the_pattern()

      # for tight upper bounds calculation
      subset_joincounts <- pattern counts

      If pattern_support > min_support
            If cross_k > ε:
                  Output_the_pattern
      else:
            prune_all_parents()
            return True

else:
      # As previously enumerated
      If pattern_support > min_support:
            return False
      else:
            prune_all_parents()
            return True

return False
```

root_enumeration:

```
If pattern already enumerated:
      return <DOUBT>
else:
      # pattern not enumerated
      count_the_pattern()

      <DOUBT> No superset count propagation

      UB_lattice <- upper_bound(leaves_joinsetcount, candidate_count)
      If UB_lattice < ε:
            return <DOUBT>

      else:
            # Upper bound lattice of root above threshold
            If pattern_support < min_support:
                  # first node without parents, no effects
                  return
            else:
                  If cross_k > ε:
                        Output_the_pattern

            # for tight upper bounds calculation
            Update supsersetcount of children
```

leaf_enumeration:

```
If pattern already enumerated:
      # for tight upper bounds calculation
      leaves_joincounts <- pattern counts

      if pattern_support < min_support:
            prune_all_parents()
            return True

else:
      # pattern not enumerated
      count_the_pattern()

      # for tight upper bounds calculation
      leaves_joincounts <- pattern counts

      if pattern_support < min_support:
            prune_all_parents()
            return True

      else:
            # support above threshold
            If cross_k > ε:
                  Output_the_pattern

return False
```

PS: The other functions are implementation dependent and do not require an overview at the moment.