Table of Contents

Co-occurrence Pattern Detection

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., $|C \bowtie W_n|$.

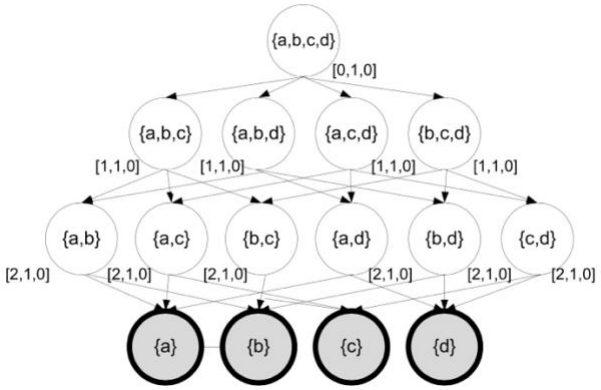
- **Local Upper Bound:** $\frac{Num\ of\ readings\ |T|}{Anom.\ Windows\ |W_n|} * \frac{Min.\ Leaf\ Joinset\ Count\ (Upper_{loc}\ |C \bowtie W_n|)}{Superset\ Pattern\ Count\ (Lower\ |C|)}$

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

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

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

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

else:
      # No pruning occurs

      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 supersetcount 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.