CSCI-8970 ADVANCED DATA INTENSIVE PROJECT REPORT

W-Shingling-based Wikipedia Evolution Study

by

Venkata Sravya Maram (811620498)

Lokesh Dananjayarao Adusumilli (811314401)

1.Introduction:

Online documents are updated at a very fast pace, and it becomes important to compare versions over time. Wikipedia pages are ideal for this type of study because of their revision history. W-shingling is a method that breaks text into continuous word sequences, called shingles, which makes it possible to calculate similarity even when edits are small or scattered.

The goal of this project is to design and implement a W-shingling system for Wikipedia city pages, compute Jaccard similarity between the latest page and older versions, and evaluate how different shingle window sizes (w) and λ-min values influence both similarity accuracy and computational cost.

2.Datasets and Preprocessing:

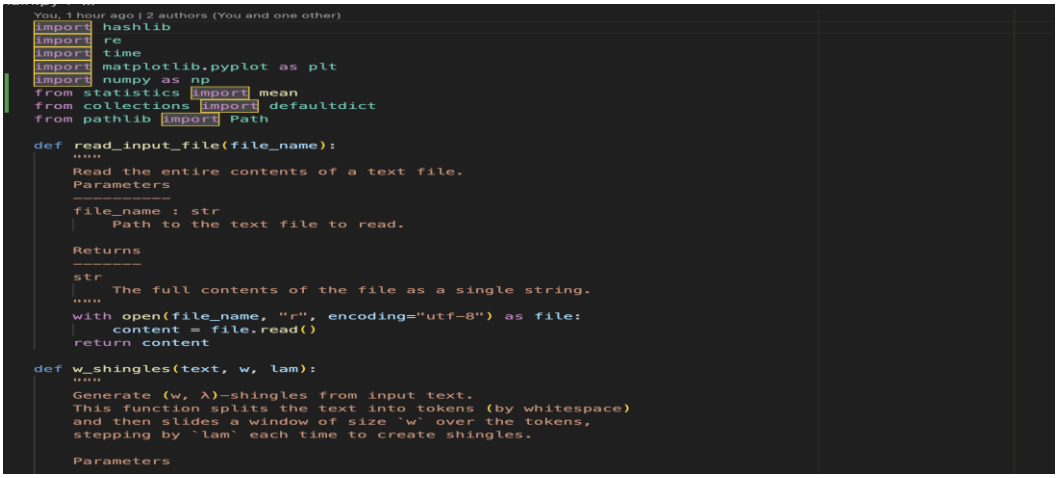
The dataset corpus was prepared collectively for the course. Each student selected four cities in the United States from the official Wikipedia list. For every city chosen, there were multiple iterations of its page: that is, the present version (WCT) and recurring historical iterations, such as VC(T-3), VC(T-6), VC(T-9), and so on until VC(T-147). This design enabled the investigation of gradual and long-term change.

The raw pages were already cleaned so that things like tables, infoboxes, and other formatting elements were removed. This ensures that only the plain text content is used when processing. Before running the algorithms, all files are converted to lowercase so that differences in capitalization do not affect the results. The system also checks that each city folder has the necessary files, and in cases where files are missing, those cities are skipped.

This preparation step makes the dataset consistent and reliable, which is important for measuring how the content of Wikipedia pages changes over time.

3.System Design:

The process is designed in sequential actions, with each action concentrating on one element of the process.



- **Shingling (w, λ):** Each document first splits into tokens (words), and a sliding window of size w moves across the text to produce shingles, small sequences of contiguous tokens that help form a representation of the local context of the page.

```python
def w_shingles(text, w, lam):
    """
    Generate (w, λ)-shingles from input text.
    This function splits the text into tokens (by whitespace)
    and then slides a window of size `w` over the tokens,
    stepping by `lam` each time to create shingles.

    Parameters
    ----------
    text : str
        The input text to shingle.
    w : int
        The window size (number of tokens per shingle).
    lam : int
        The step size between consecutive shingles.

    Returns
    -------
    list of tuple
        Each tuple contains `w` consecutive tokens.
    """
    # Tokenize based on spaces
    tokens = text.split()
    shingles = []

    # Slide window of size w across tokens with step lam
    for i in range(0, len(tokens) - w + 1, lam):
        shingle = tuple(tokens[i:i + w])  # make tuple
        shingles.append(shingle)

    return shingles
```

- **Hashing:** The shingles then undergo the MD5 hash process, which produces fixed-length hash values to facilitate comparisons among documents with different lengths.

```python
def hash_shingles(shingles):
    """
    Convert shingles to MD5 hashes.
    Each shingle is assumed to be a tuple of tokens (strings).
    The function joins the tokens with a separator, encodes to bytes,
    and computes an MD5 hash for each shingle.

    Parameters
    ----------
    shingles : list of tuple
        List of shingles, where each shingle is a tuple of tokens.

    Returns
    -------
    list of str
        List of MD5 hash strings corresponding to each shingle.
    """
    hashes = []
    for shingle in shingles:
        # Join tokens using a separator; you can lowercase here if desired
        shingle_str = " ".join(shingle)  # skip .lower() if you care about case
        # Encode to bytes and hash using MD5
        shingle_hash = hashlib.md5(shingle_str.encode("utf-8")).hexdigest()
        hashes.append(shingle_hash)

    return hashes
```

- **λ-min Selection**: Finally, from the complete set of hashes only the (lexicographically) minimum λ hash values are retained. If $\lambda = \infty$, then all hashes will be retained. This selects only a sample of the overall hashes and creates a representation of the document that reduces the quantity of each document's hash values while offering enough information about the document for comparisons.

```
def _lambda_min(hashes, lam):
    """
    λ-min selection on hashed shingles.
    Keep the lexicographically smallest λ hash strings to form a compact
    sketch of the document. If λ is ∞, keep all hashes.
    """
    if lam == float("inf"):
        return list(hashes)
    lam = int(lam)
    return sorted(hashes)[:lam]
```

- **Jaccard Similarity**: To assess the similarity factor between two versions, the Jaccard similarity is calculated by measuring the size of the intersection divided by the size of the union of the respective hash sets. This similarity score measures the degree of similarity between the two versions in content.

```
def _jaccard(hashes_a, hashes_b):
    """
    Jaccard similarity between two hash sets.
    """
    A, B = set(hashes_a), set(hashes_b)
    if not A and not B:
        return 1.0
    return len(A & B) / len(A | B) if (A or B) else 0.0
```

- **Corpus Handling:** Every city folder contains several text files. The system automatically detects which of those files is the current version and, furthermore, identifies the older versions based on the naming pattern. Therefore, the comparison between the two versions includes all previous versions for each city.

```python
def _find_city_versions(city_dir: Path):
    """
    Locate the current file and all older versions for a city.
    Supports both naming schemes:
      1) <City_State>_C.txt , <City_State>_C-3.txt , ...
      2) VC_T.txt , VC_T-3.txt , ...
    Returns (current_path, [(version_label, path, lag_int), ...]) sorted by lag.
    """
    files = [p for p in city_dir.iterdir() if p.is_file() and p.suffix == ".txt"]
    current = None
    older = []

    for p in files:
        if _PAT_C_CURR.match(p.name) or _PAT_VC_CURR.match(p.name):
            current = p
            break
    if not current:
        raise FileNotFoundError(f"Missing current file in {city_dir} (need *_C.txt o

    for p in files:
        m1 = _PAT_C_OLD.match(p.name)
        m2 = _PAT_VC_OLD.match(p.name)
        if m1:
            lag = int(m1.group(2)); older.append((f"C-{lag}", p, lag))
        elif m2:
            lag = int(m2.group(1)); older.append((f"VC_T-{lag}", p, lag))
```

- **Timing Measurement:** The system measures the time it takes to process all documents for each pair of (w, λ) values. To provide a reliable measurement, the experiment is run multiple times (3 times), and the average time is reported.

```python
def _timing_over_corpus(cities, w_values, lam_values, root: Path, out_dir: Path, runs=3):
    """
    For each (w, λ), measure average time (seconds) of:
      shingling (stride=1) → hashing → λ-min selection,
    aggregated across all documents (all cities, all versions).
    Writes: output/results/timing.csv with columns w,lambda,avg_seconds
    """
    texts = _collect_all_texts(root, cities)
    out_dir.mkdir(parents=True, exist_ok=True)
    csv_file = out_dir / "timing.csv"

    with open(csv_file, "w", encoding="utf-8") as f:
        f.write("w,lambda,avg_seconds\n")
        for w in w_values:
            for lam in lam_values:
                times = []
                for _ in range(runs):
                    t0 = time.perf_counter()
                    for tx in texts:
                        sh = w_shingles(tx, w, 1)
                        hh = hash_shingles(sh)
                        _ = _lambda_min(hh, lam)
                    times.append(time.perf_counter() - t0)
                lam_str = "inf" if lam == float("inf") else str(lam)
                f.write(f"{w},{lam_str},{sum(times)/len(times):.6f}\n")
    print(f"[timing] results → {csv_file}")
```

4. Experimental setup: The following *(w, λ)* pairs were tested as required:

- *(25, 8), (25, 16), (25, 32), (25, 64), (25, ∞)*
- *(50, 8), (50, 16), (50, 32), (50, 64), (50, ∞)*

Three main experiments were conducted:

1. **Similarity analysis:** Jaccard similarity between the current version and older revisions.
2. **Approximation check:** Identifying λ values that produced results closest to $\lambda = \infty$.
3. **Efficiency study:** Measuring runtime of shingle generation for the entire corpus at each parameter setting.

Each experiment was repeated multiple times, and average values were reported to account for runtime variation.

5. Implementation:

The code deals with plain-text files for each city located in the data/ (or Data/) directory, determining both the current version and its historical versions by the names _C.txt (current) and VC_T.txt (historical, where T represents the version lag). The contents of the file would be converted to lowercase, tokenized by whitespace, and shingled via a fixed stride of 1. For the window sizes w □ {25,50}, the shingles would be hashed using MD5, creating a fixed-length annotated identifier. To improve the efficiency of the code, the set of hashes for the current version would be computed just once per window size and employed for every sketch size. A λ-min sketch is built from those hashes by selecting the lexicographically smallest λ hash values where $\lambda \in \{8,16,32,64, \infty\}$; the case of $\lambda = \infty$ would keep all hashes without sketching them. Plots are shown in the last of the report.

For every city and every specified window size w, the Jaccard similarity is calculated from the current version's sketch to the sketch of every historical version. The historical versions are aligned along the x-axis according to their numeric lag taken from the filenames to reflect the temporal sequence of the documents. While results are saved in CSV format, the pipeline will generate a single PNG image for every city at every window size in output/plots/. Each PNG image will contain 5 vertically stacked subplots—one for each λ—visualizing how the Jaccard similarity changes over time for the current version of the document. The plots allow for easy comparisons of sketch similarity change across different amounts of sketch compression and sketch.

5. Observations:

The closest-λ analysis showed that using a smaller, or "reduced," set of shingles can closely approximate the results of using all shingles (the $\lambda = \infty$ baseline) without losing much accuracy. For most cities and for both window sizes, the λ values of 32 and 64 were consistently the closest finite values to $\lambda = \infty$. This suggests that these values are large enough to capture a good variety

of shingles while still avoiding the high computational cost of using all of them. Smaller λ values, specifically 8 and 16, were found to be less accurate and resulted in a wider gap from the baseline.

Here, the closest λ value consistently turned out to be **64** in many cases. For cities with longer articles and frequent revisions, $\lambda = 64$ provided results very close to the $\lambda = \infty$ baseline, making it the most effective choice. This indicates that the closest λ depends on both the length of the text and the frequency of edits, with $\lambda = 64$ emerging as the most reliable setting. Such behavior suggests that adapting λ dynamically—while often favoring 64—could balance efficiency and accuracy across different contexts.

Sample λ values that are identified as closest to ∞ are tabulated below:

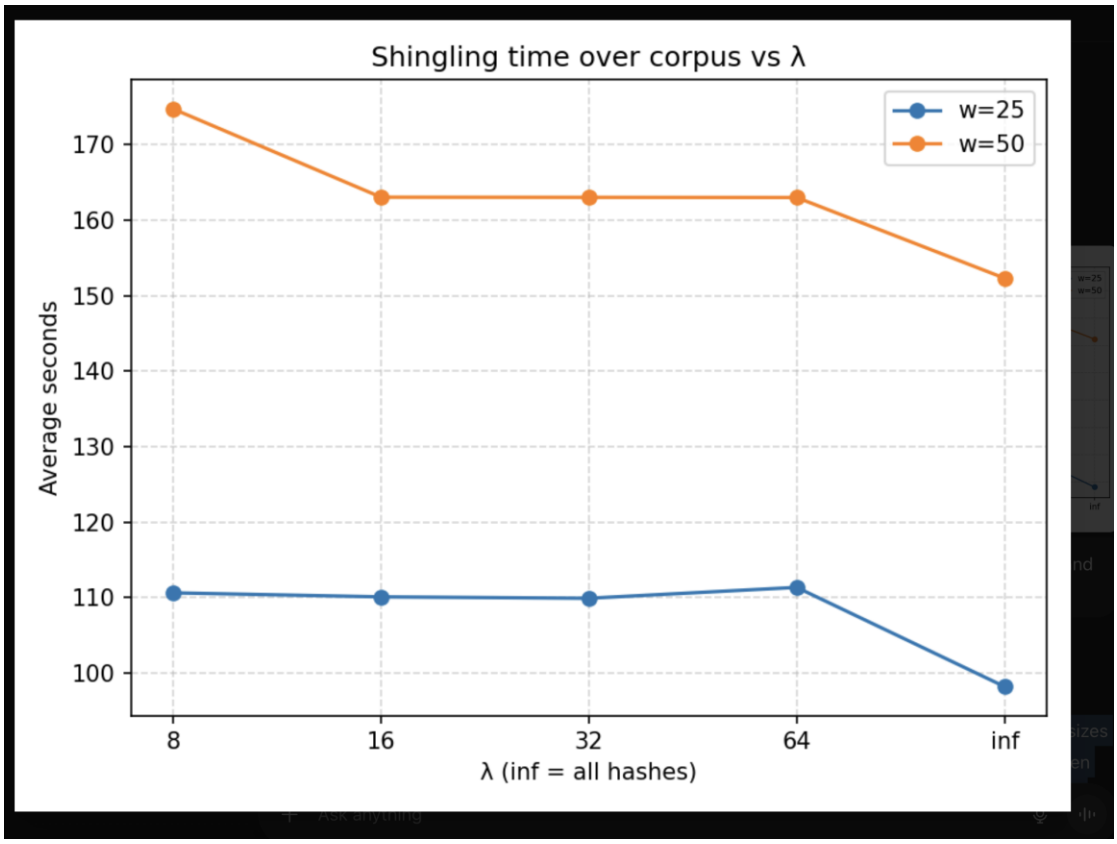
Berkeley_CA

w	Closest_lambda	abs_gap_vs_inf	inf_mean	lambda_mean
25.0	64.0	0.020042	0.846495	0.826453
50.0	64.0	0.002781	0.765801	0.768582

Alpharetta_GA

w	Closest_lambda	abs_gap_vs_inf	inf_mean	lambda_mean
25.0	64.0	0.05842	0.35472	0.2963
50.0	64.0	0.021393	0.271147	0.249755

The timing plot demonstrates how the average shingling time varies at each of the different values of λ, for the cases of window sizes 25 and 50. For $w = 25$, the timing hovers around 110 seconds, and remains more or less stable across the λ range from 8 to 64, before dropping significantly at $\lambda = \infty$ (all hashes) to just under 100 seconds. For $w = 50$, the shingling time begins at a substantial 175 seconds, and is relatively reduced when $\lambda = \infty$ to around 152 seconds. In sum, the larger windows ($w = 50$) always take longer than the shorter ($w = 25$) window sizes. The timing plot is given below.



6.Results:

The closest-λ CSV outputs confirm that **λ = 64** consistently emerges as the best approximation across both w=25 and w=50. The absolute gap to the λ = ∞ mean similarity is typically below 0.02, showing that λ = 64 achieves near-baseline accuracy. These results demonstrate that λ-min hashing with λ = 64 provides an effective trade-off: significantly reducing computational cost while maintaining high-quality similarity estimation across revisions.

The findings show that shingle efficiency novelly improves as λ increases, as there are fewer hash operations to maintain with larger sketch sizes or the full set at λ = ∞. The value of constant decreasing average seconds for w = 50 is also indicative that λ-min sketching can find a balance between accuracy and processing costs. Lastly, the sharp decrease for w = 25 at λ = ∞, indicates that when we can maintain all our hashes, it can be faster than maintaining smaller sketches. This shows that window size and λ can both play a role in performance, although w = 25 remains the best option overall.

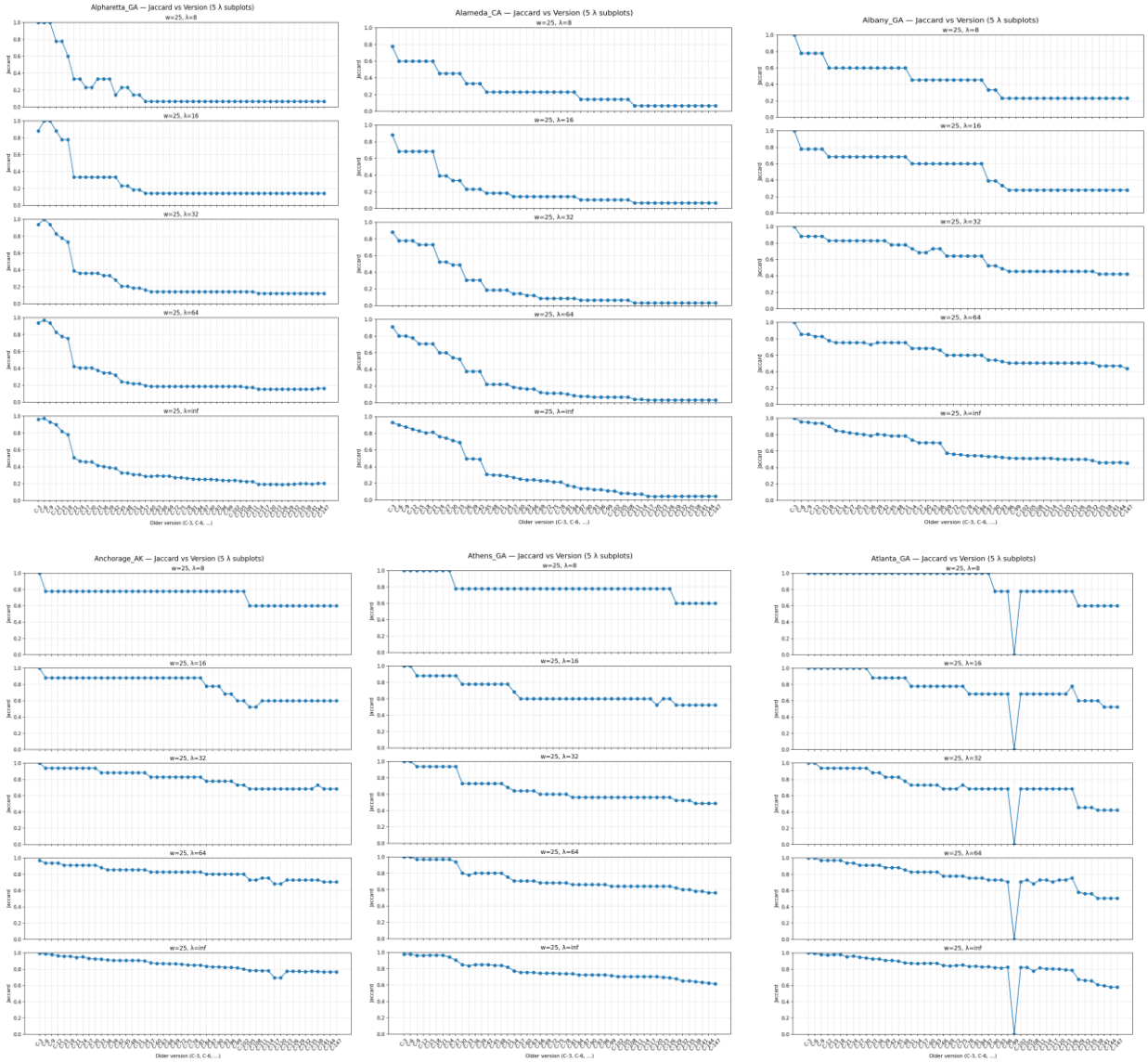
7.Conclusion:

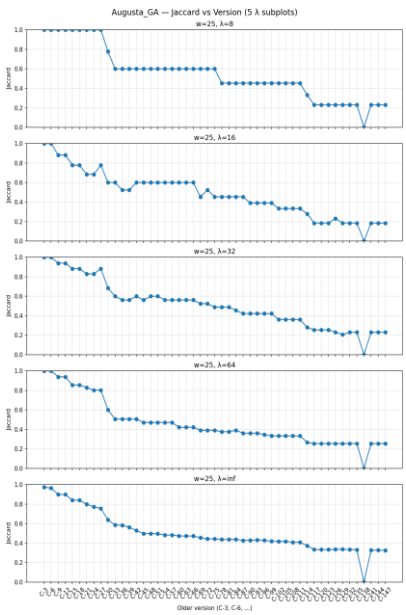
This project compares Wikipedia revisions for different cities by estimating text similarity using λ-min sketches. It splits each document into shingles (25 or 50 words long), hashes them with MD5, and then keeps only the smallest λ hashes to form a compact sketch. The full hash set is

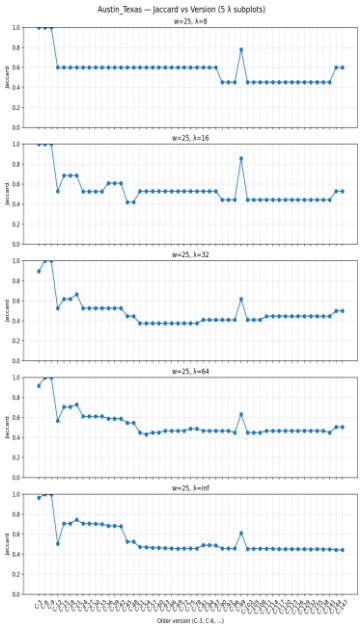
computed once per window size and reused across all λ values (8, 16, 32, 64, ∞), saving time. Results show that λ=32 (for w=25) and λ=64 (for w=50) closely match the accuracy of using all hashes (∞) while running much faster.

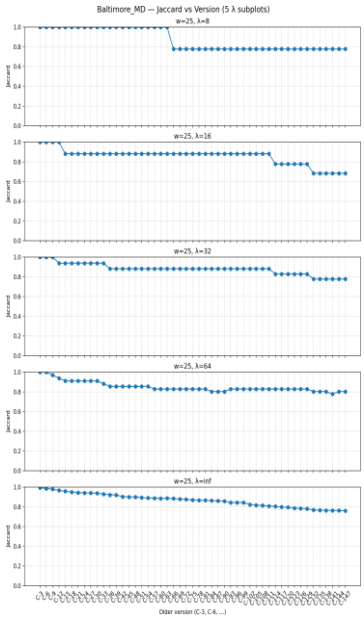
The method also reveals real editing patterns—like when a page is reverted to an older version, which shows up as a spike in similarity at a specific time lag. Versions are ordered chronologically using numbers from filenames, and results are visualized in one PNG per city and window size, with five subplots (one per λ) for easy comparison. CSVs are saved for the closest λ values, keeping the process efficient and focused on clear, scalable insights.

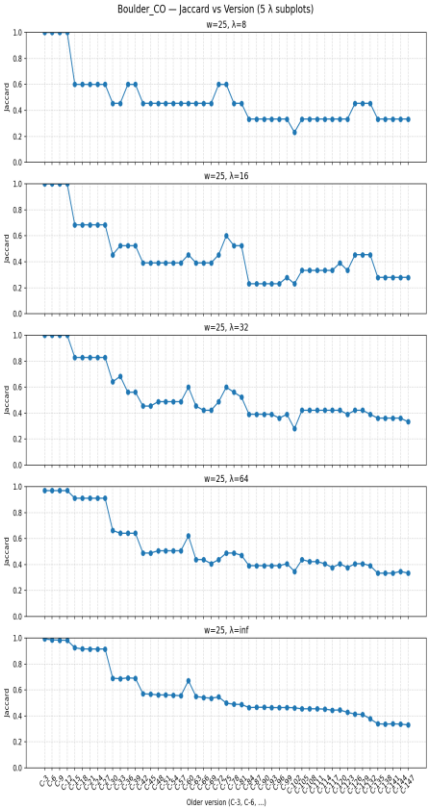
8. Graphs or Plots

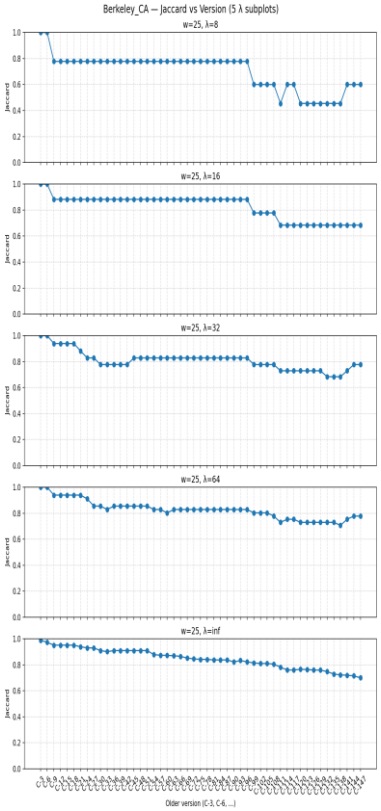


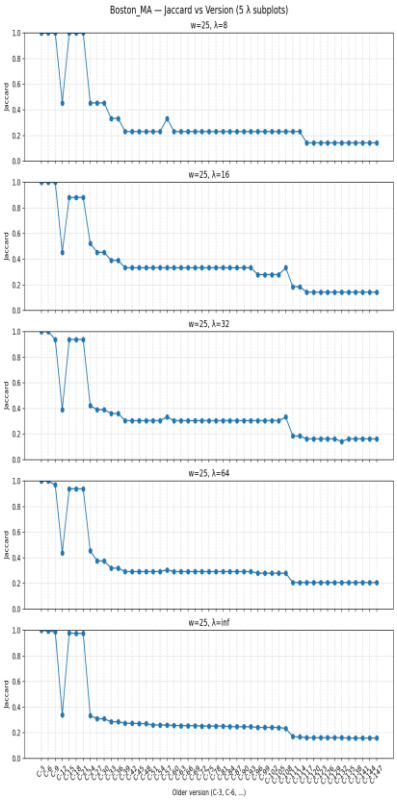

Augusta_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

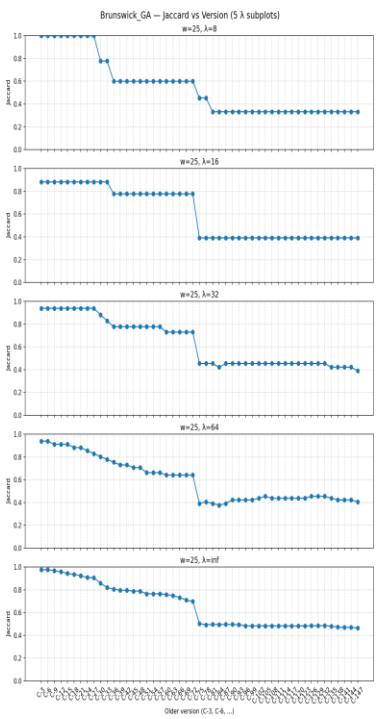

Austin_Texas — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

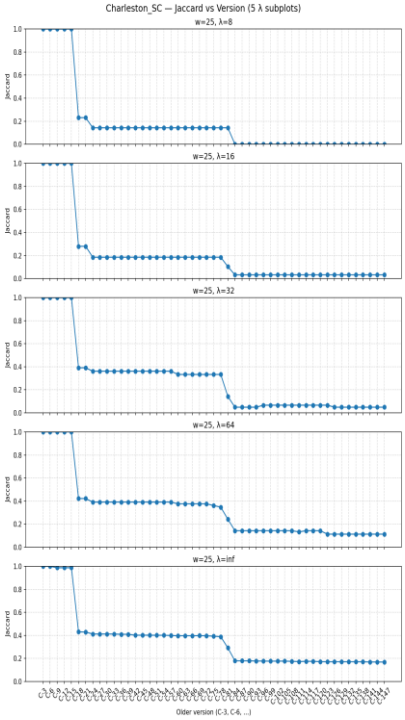

Baltimore_MD — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

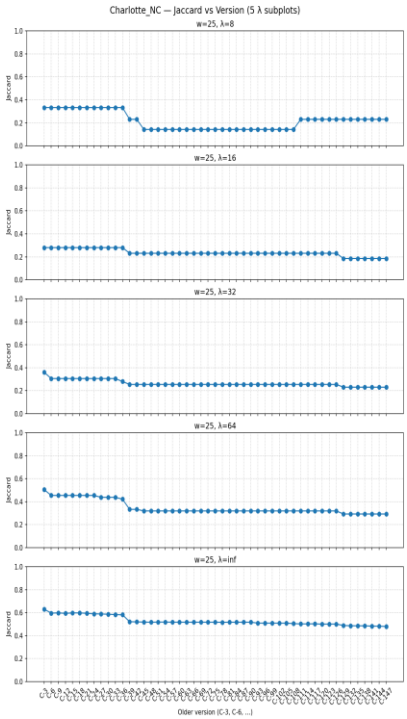

Boulder_CO — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

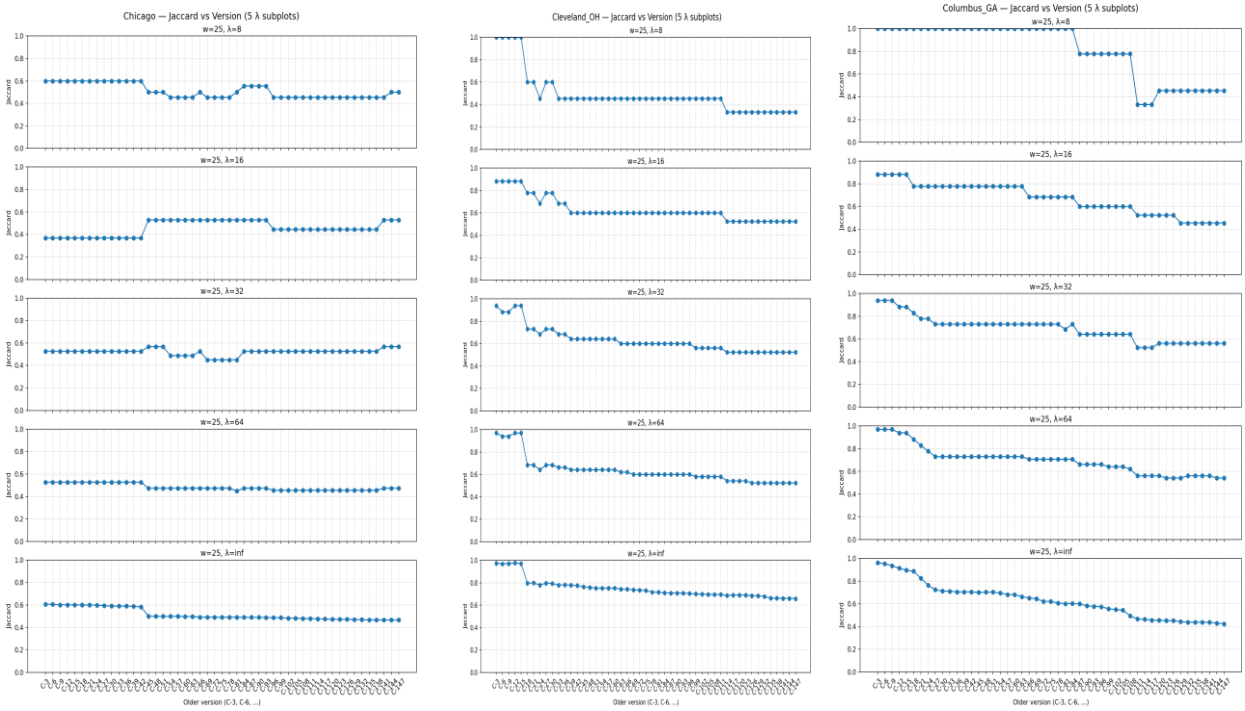

Berkeley_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Boston_MA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Brunswick_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

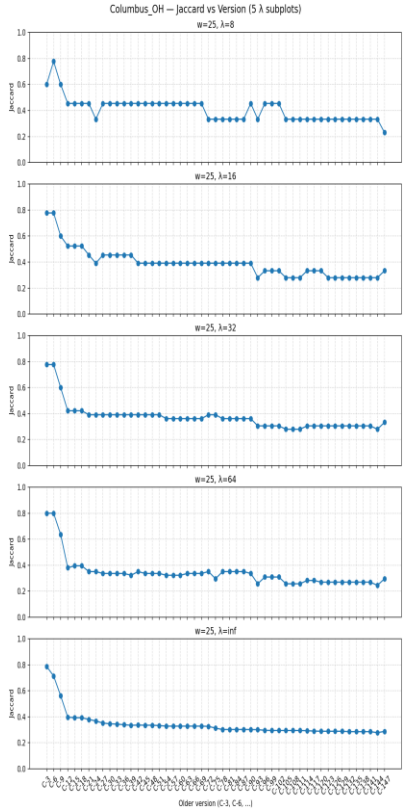

Charleston_SC — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

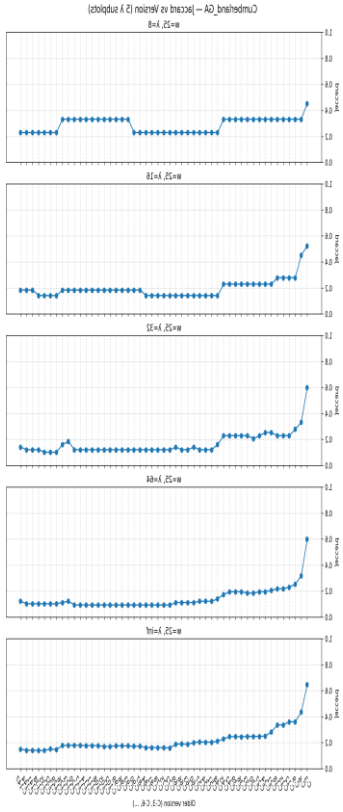

Charlotte_NC — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

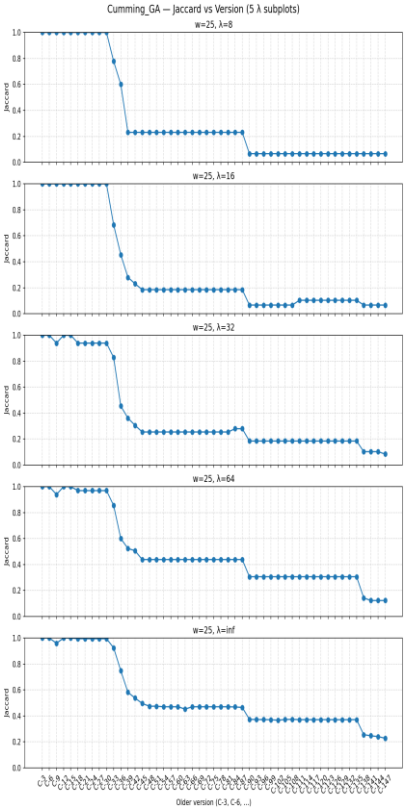

Chicago — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

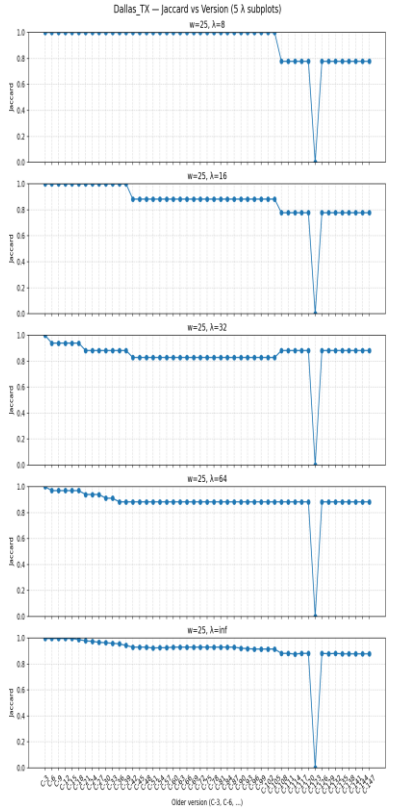
Cleveland_OH — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

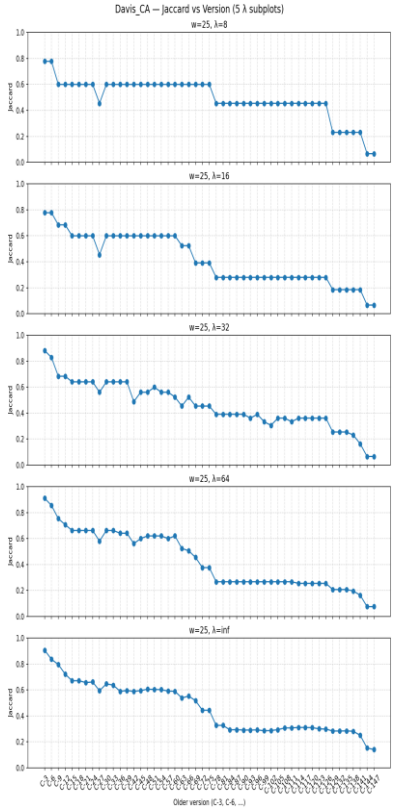
Columbus_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

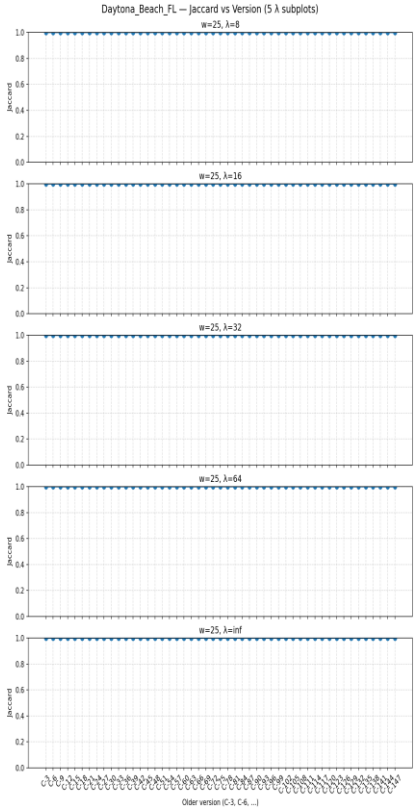

Columbus_OH — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

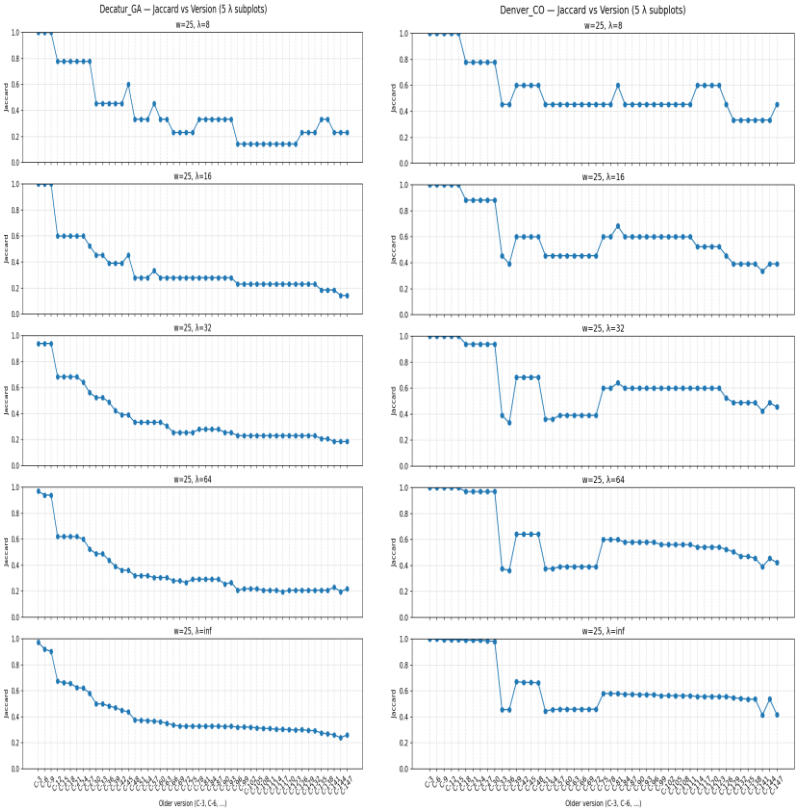

Cumberland_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Cumming_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

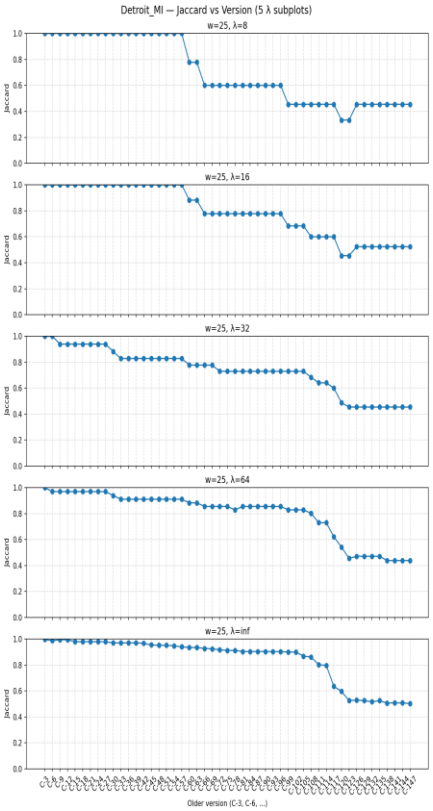

Dallas_TX — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

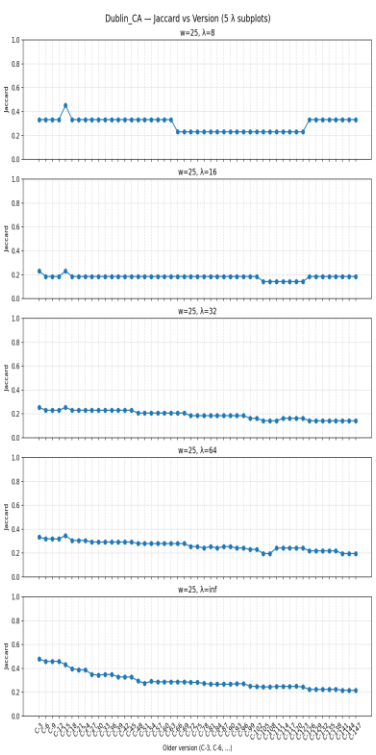

Davis_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

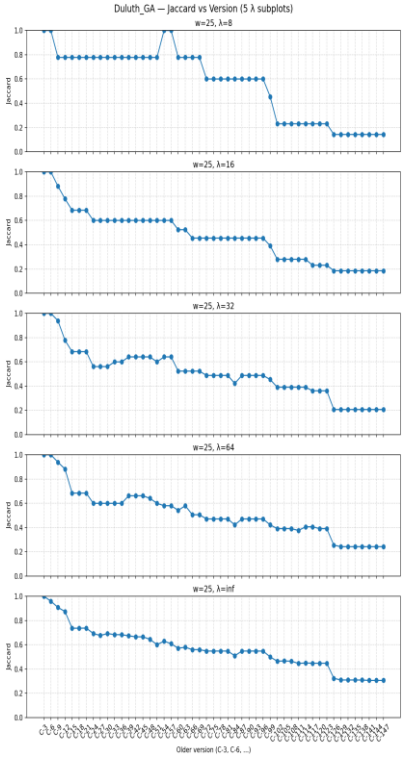

Daytona_Beach_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

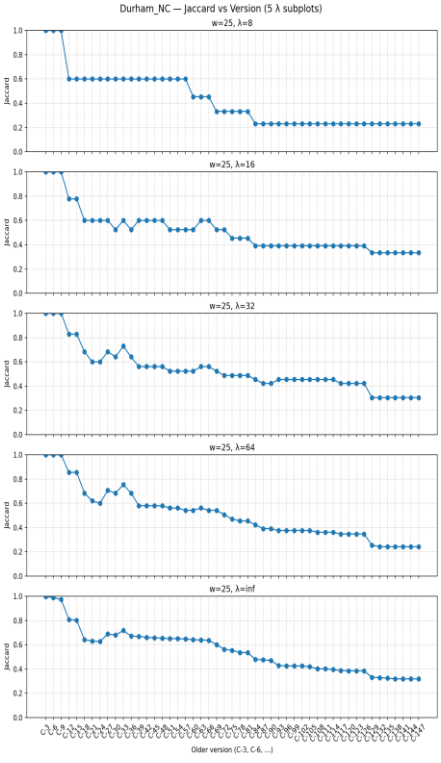

Decatur_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-1, C-6, ...)

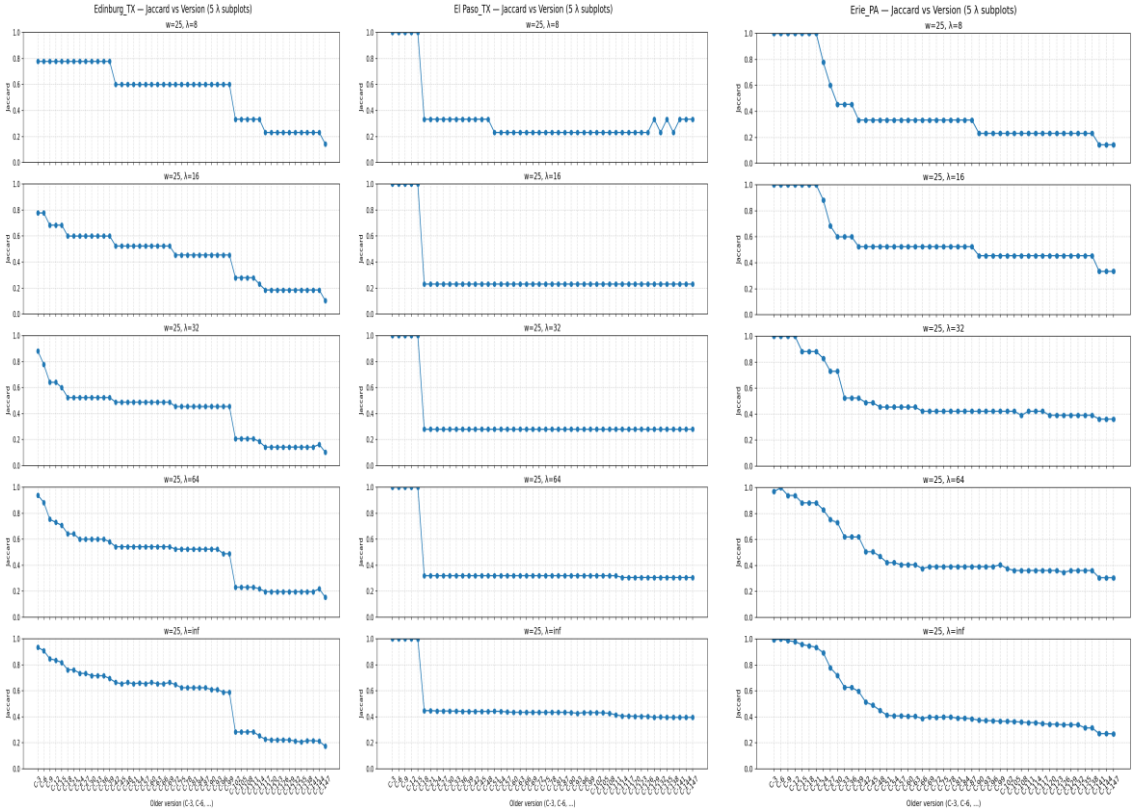
Denver_CO — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Detroit_MI — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Dublin_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

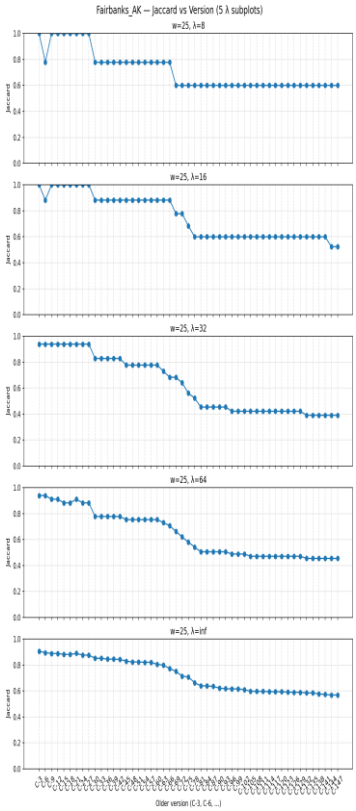

Duluth_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

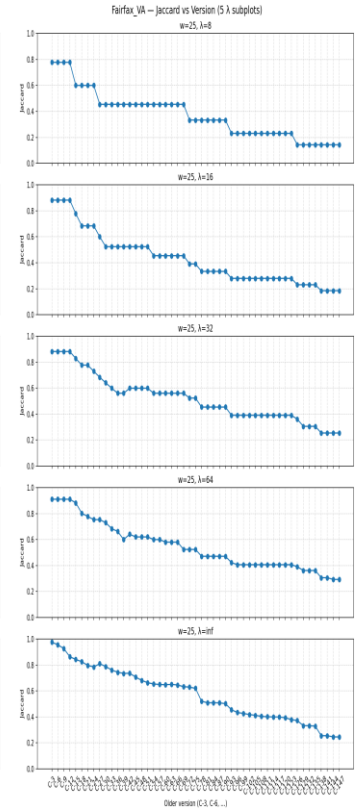

Durham_NC — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

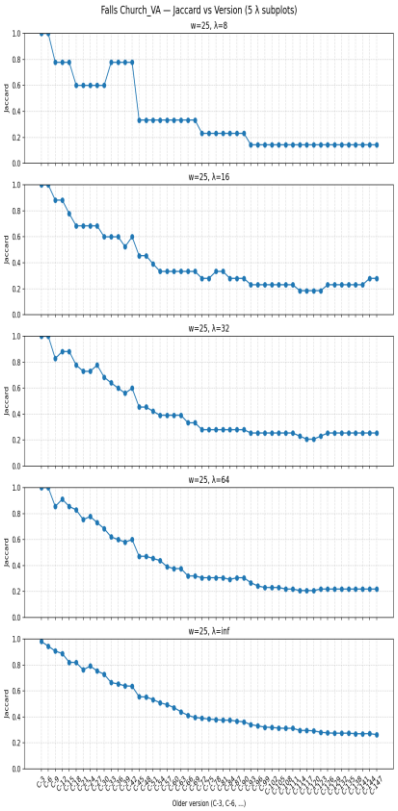

Edinburg_TX — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

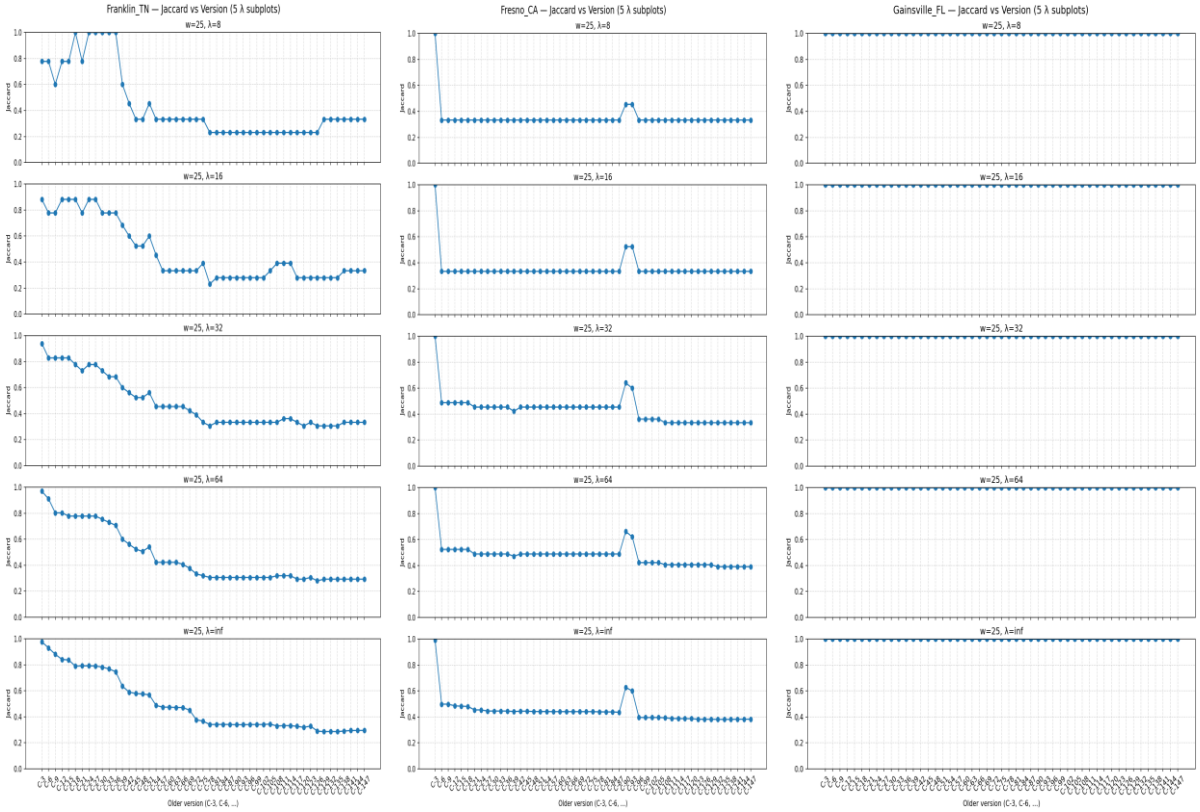
El Paso_TX — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

Erie_PA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Fairbanks_AK — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

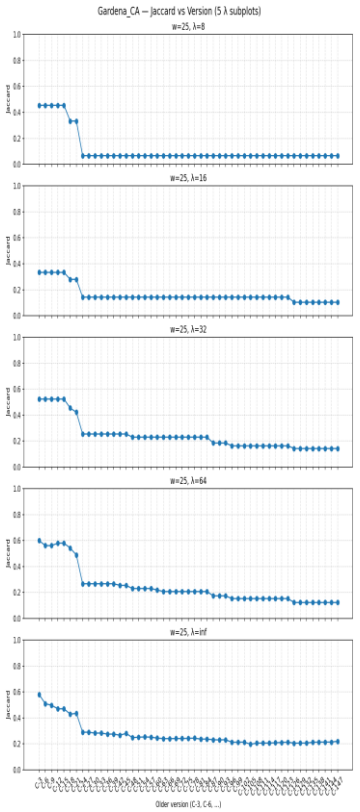

Fairfax_VA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

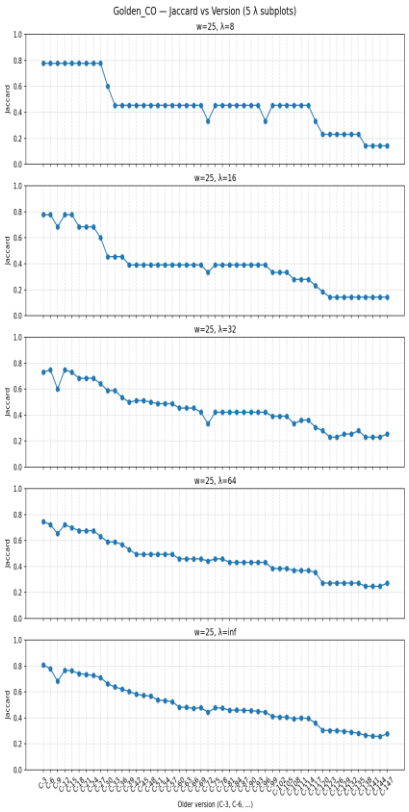

Falls Church_VA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

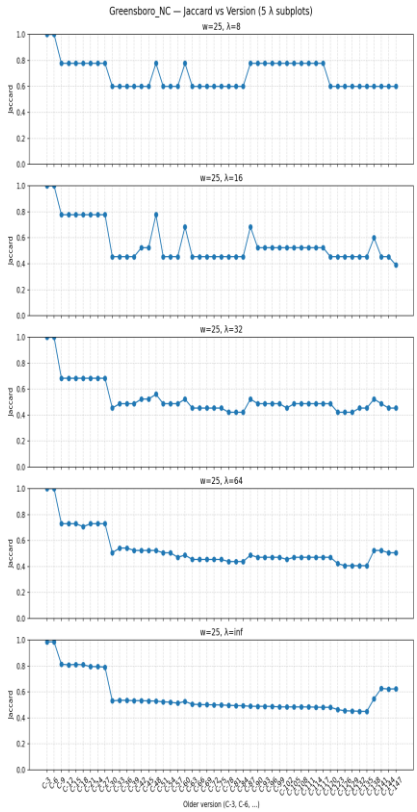

Franklin_TN — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

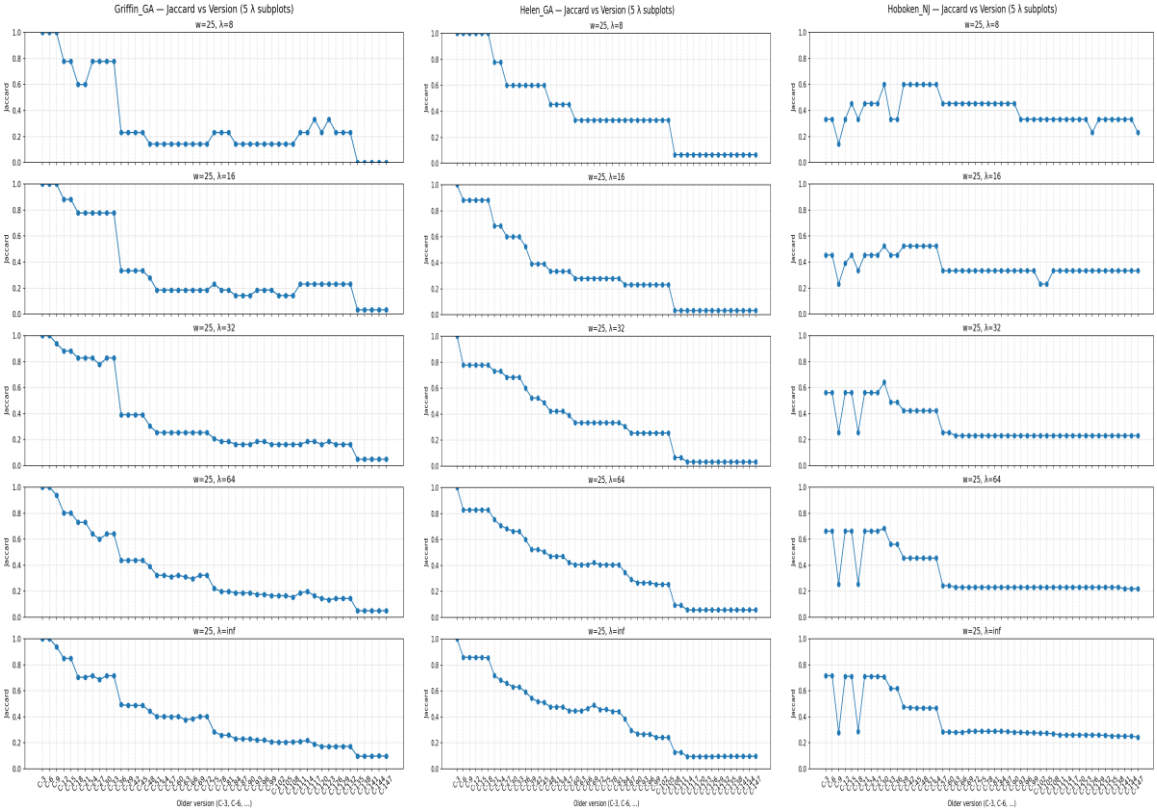
Fresno_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

Gainsville_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Gardena_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

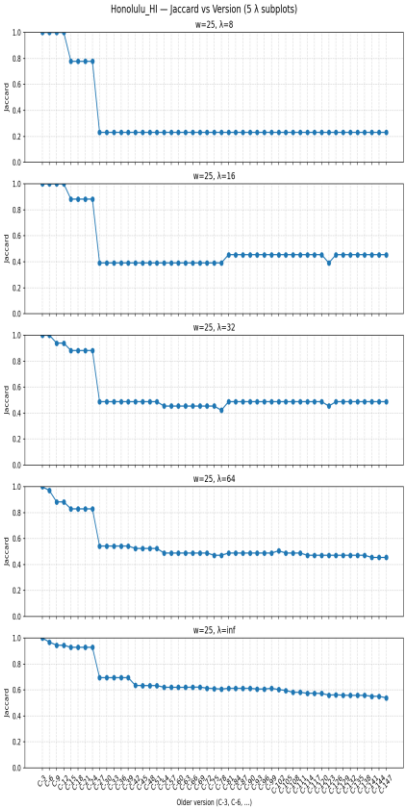

Golden_CO — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

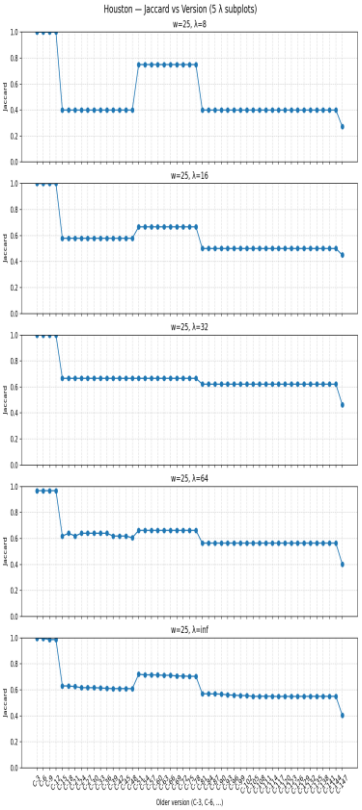

Greensboro_NC — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

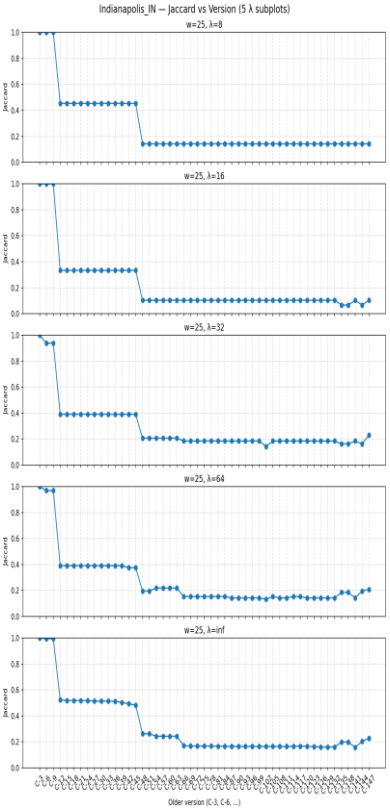

Griffin_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

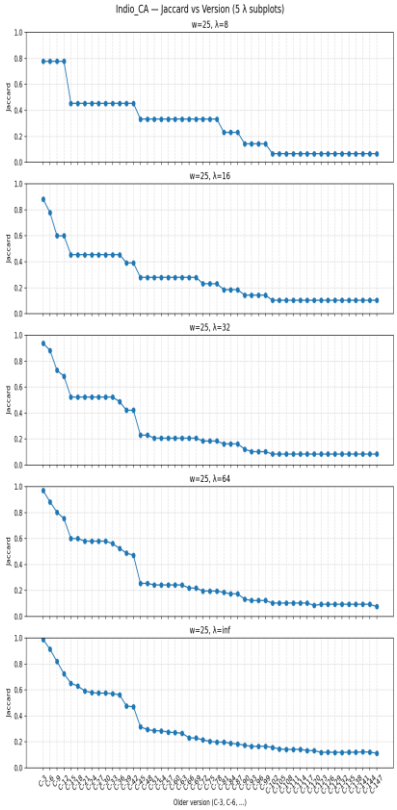
Helen_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

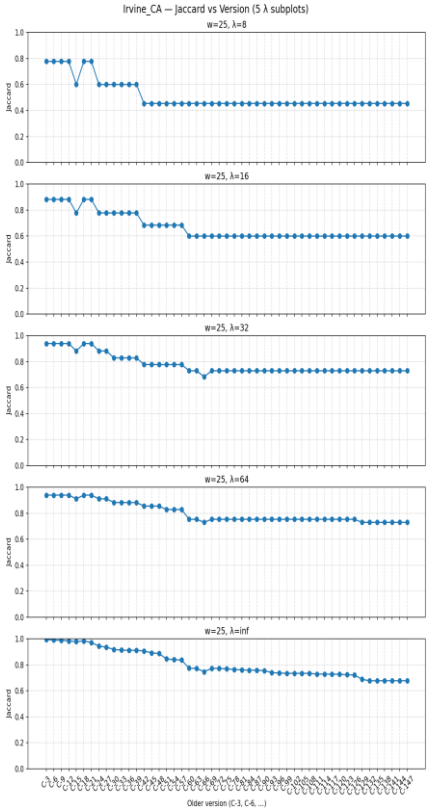
Hoboken_NJ — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

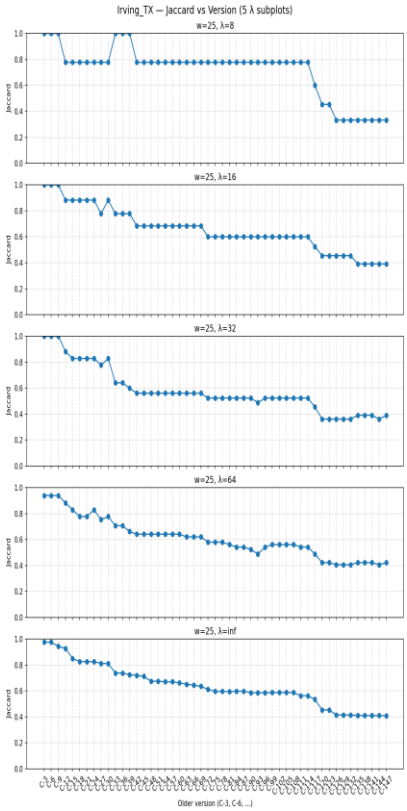

Honolulu_HI — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

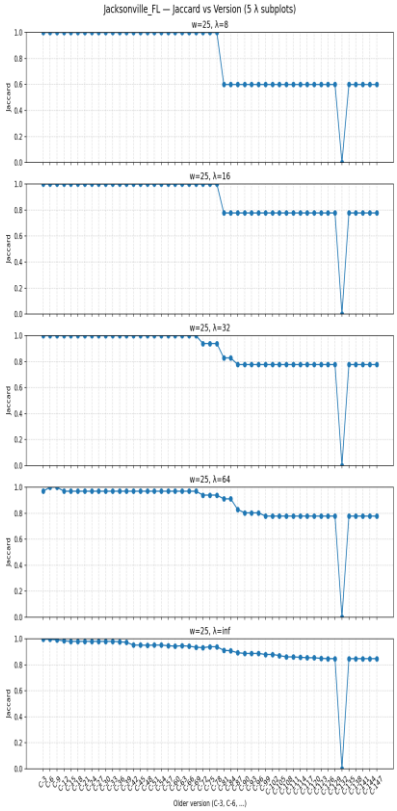

Houston — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

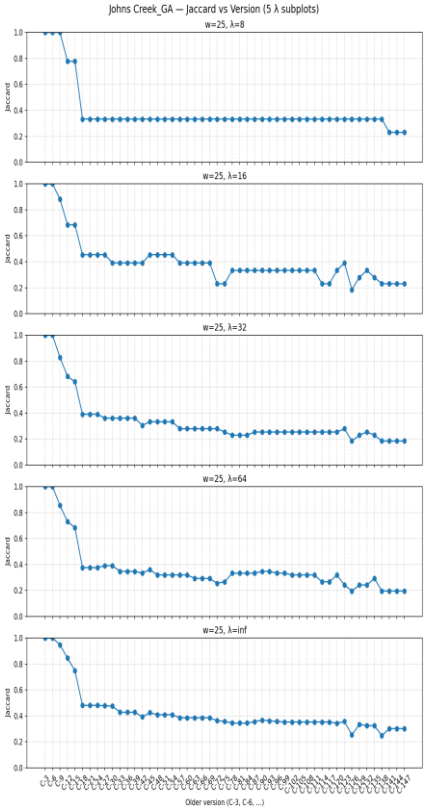

Indianapolis_IN — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

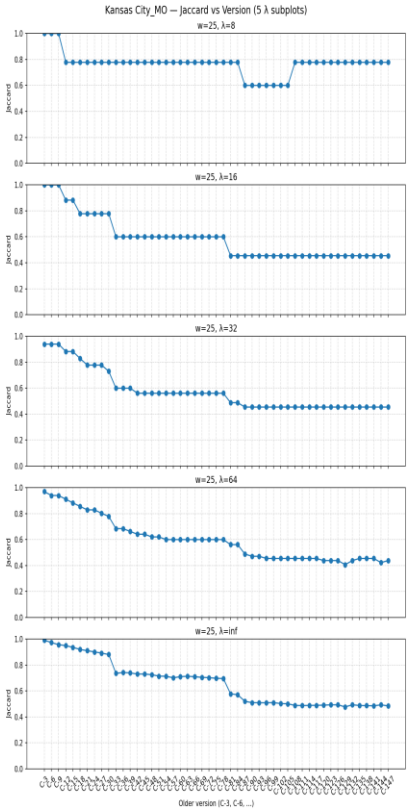

Indio_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

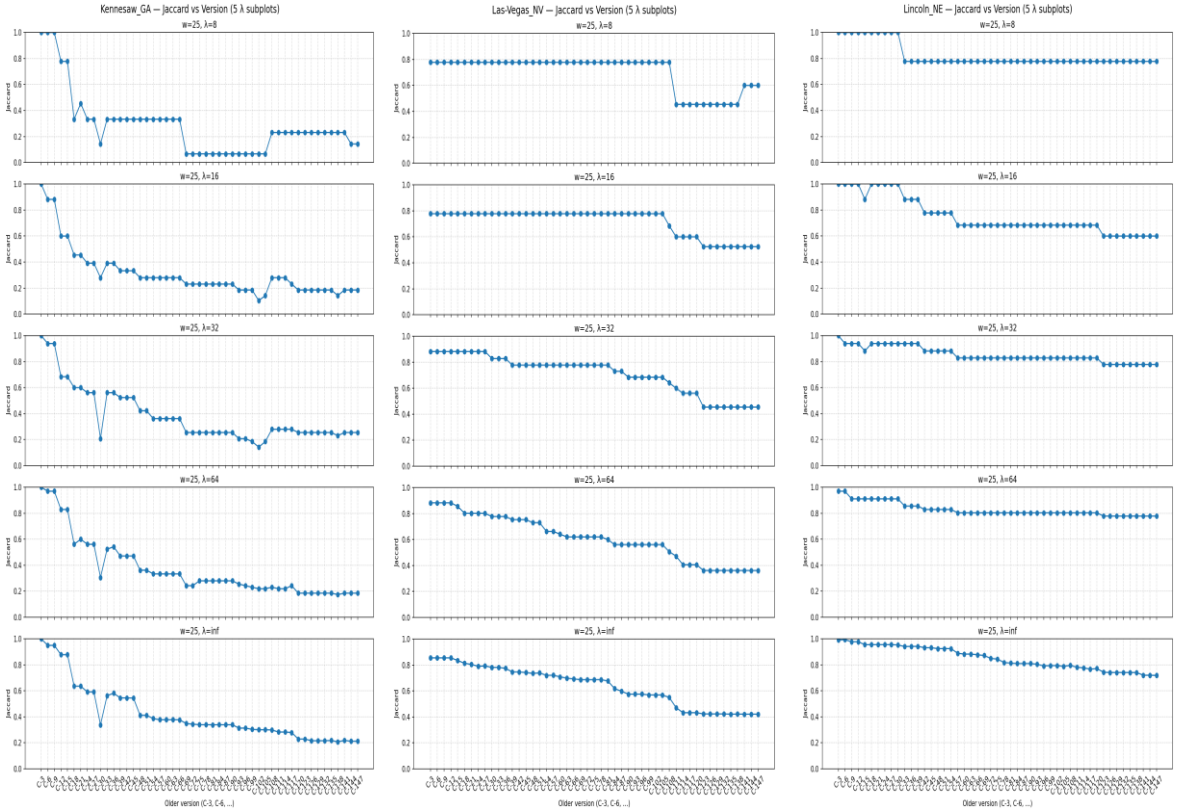

Irvine_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Irving_TX — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Jacksonville_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

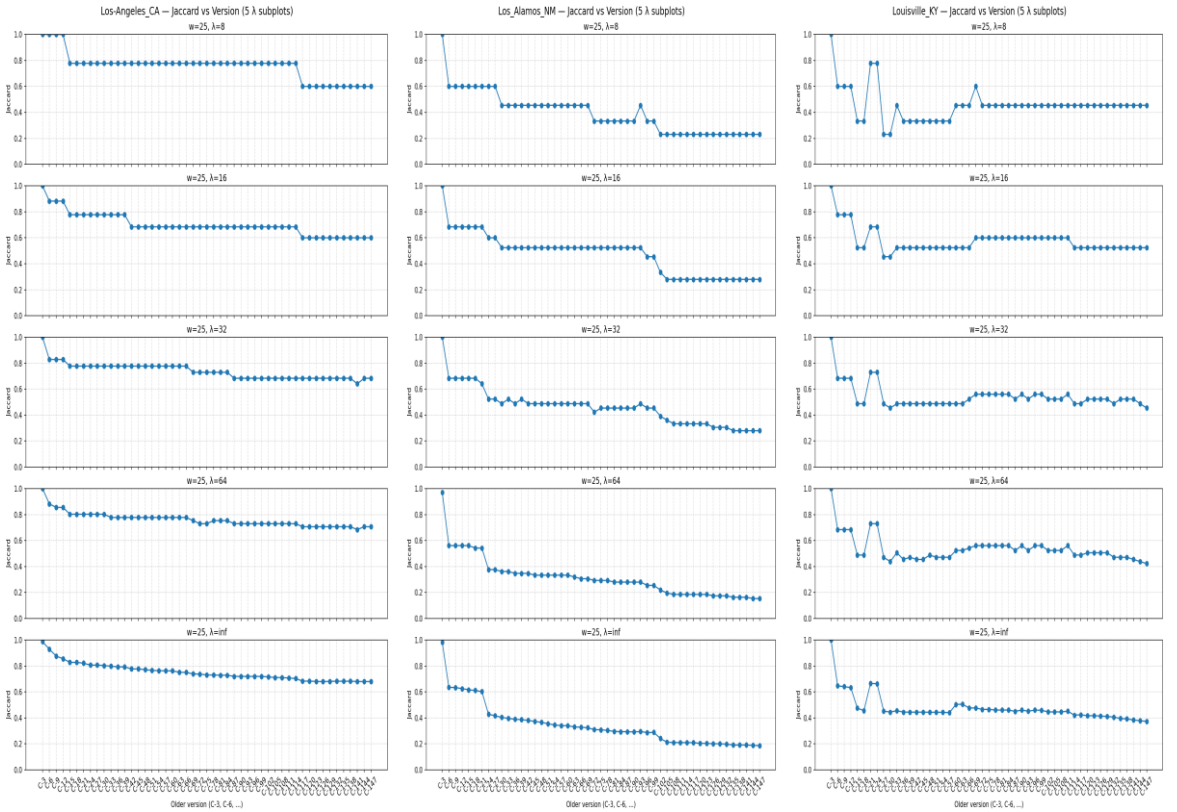

Johns Creek_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Kansas City_MO — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Kennesaw_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

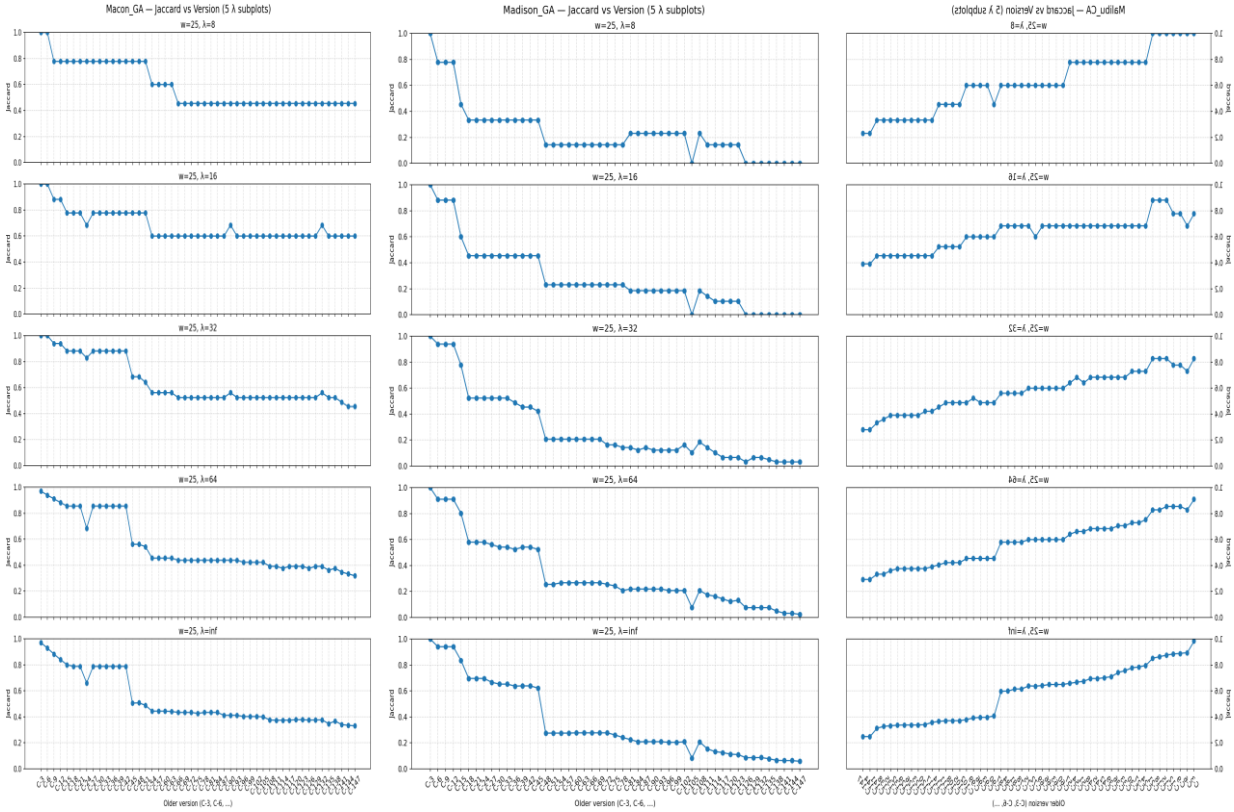
Las-Vegas_NV — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

Lincoln_NE — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)


Los-Angeles_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

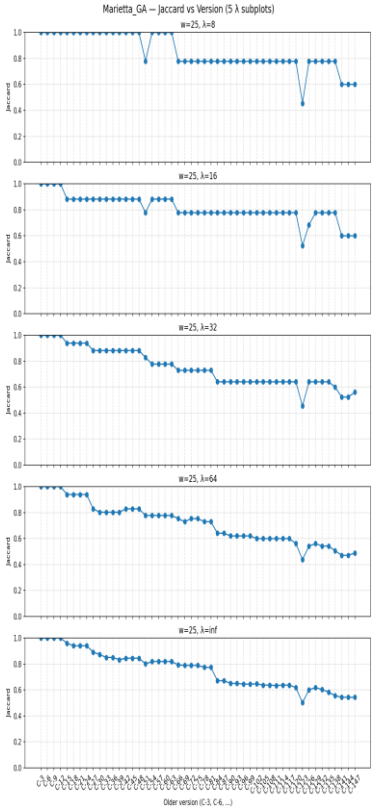
Los_Alamos_NM — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

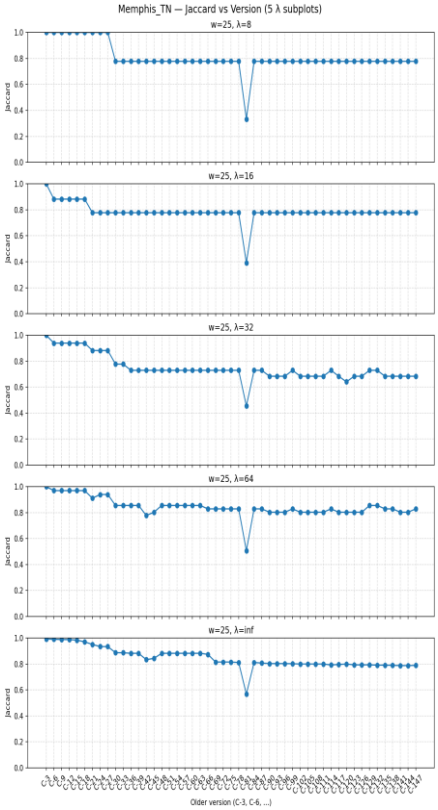
Louisville_KY — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, …)

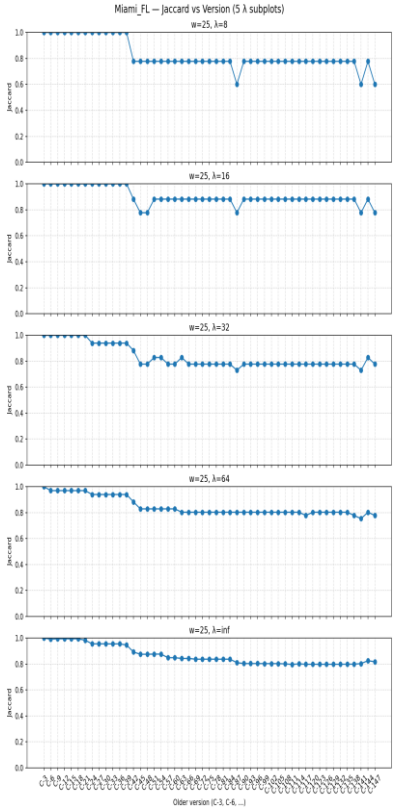

Macon_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

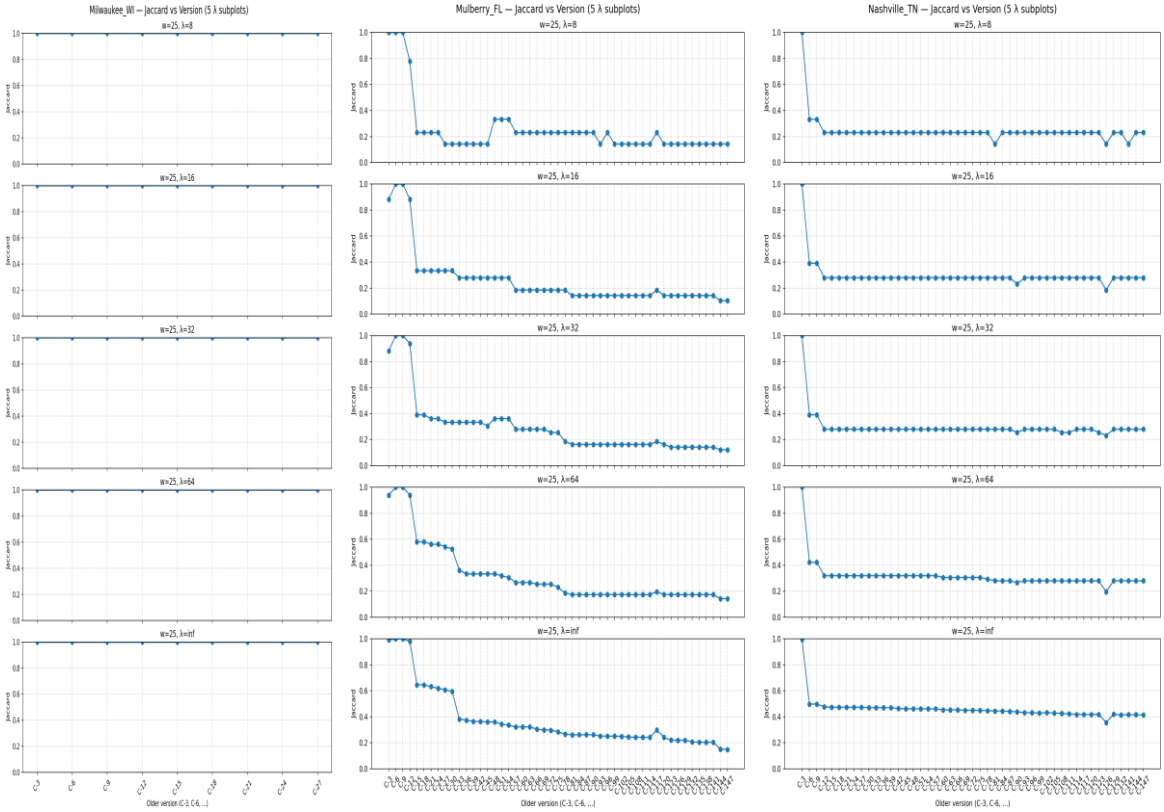
Madison_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

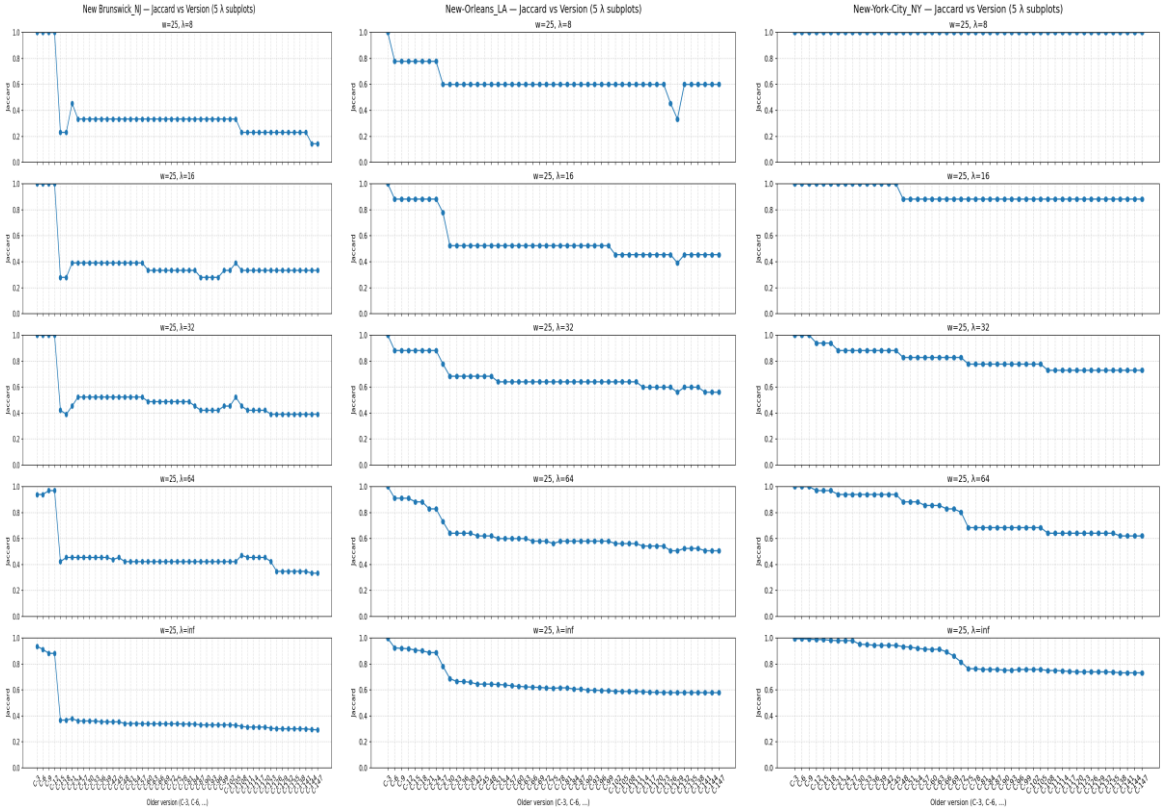
Mallipu_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

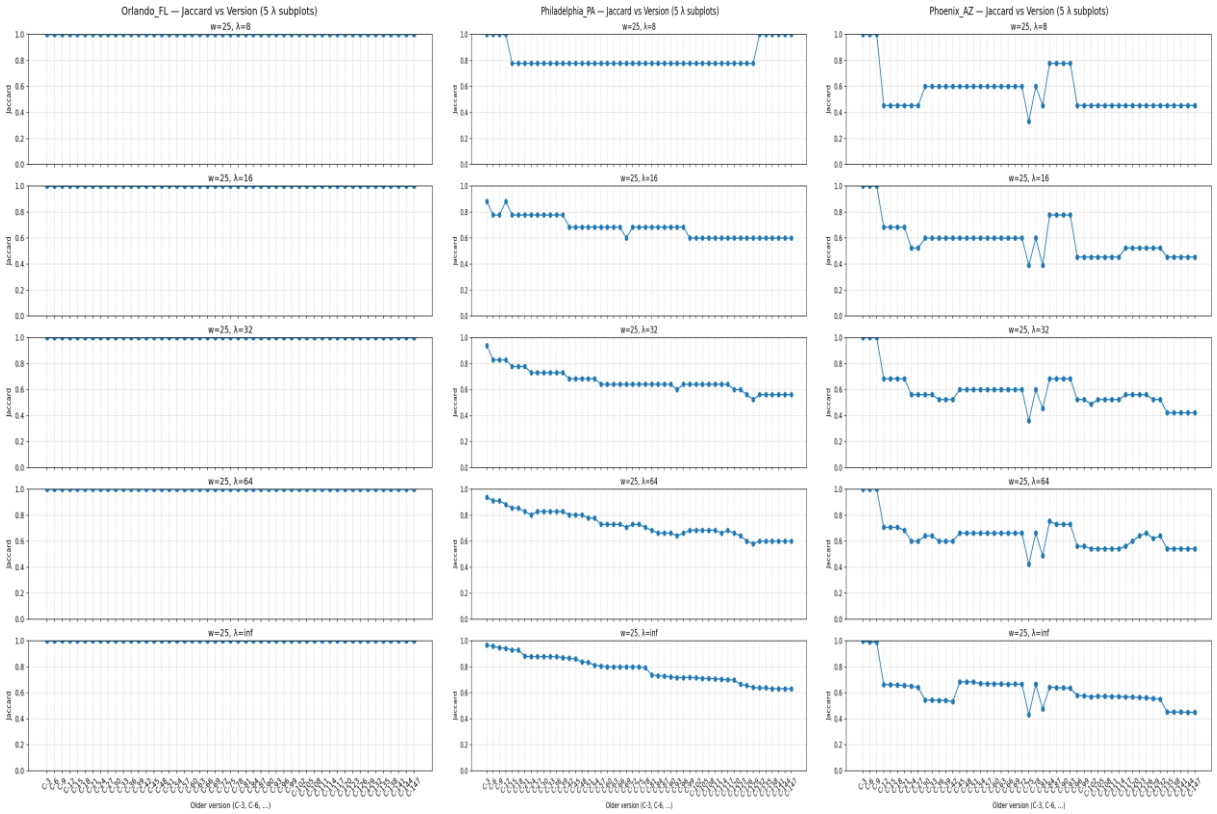

Marietta_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Memphis_TN — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Miami_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

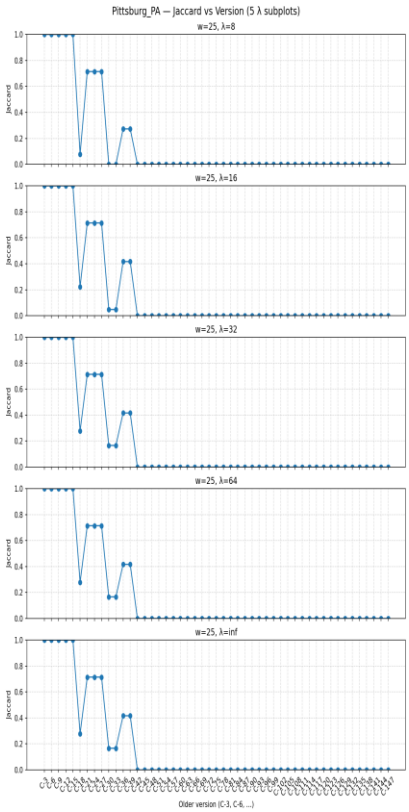

Milwaukee_WI — Jaccard vs Version (5 λ subplots)
Mulberry_FL — Jaccard vs Version (5 λ subplots)
Nashville_TN — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

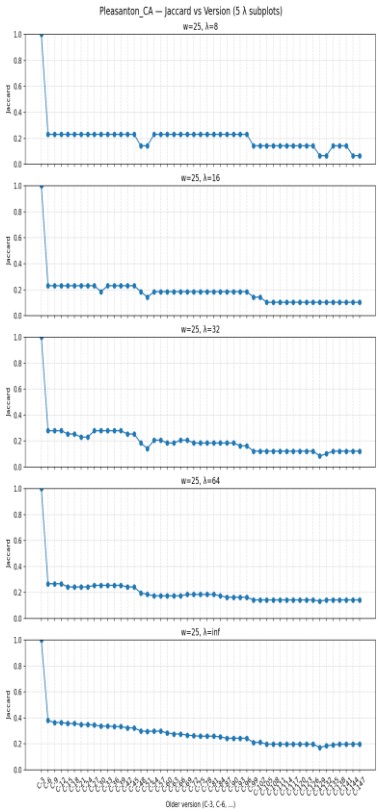

New Brunswick_NJ — Jaccard vs Version (5 λ subplots)
New-Orleans_LA — Jaccard vs Version (5 λ subplots)
New-York-City_NY — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

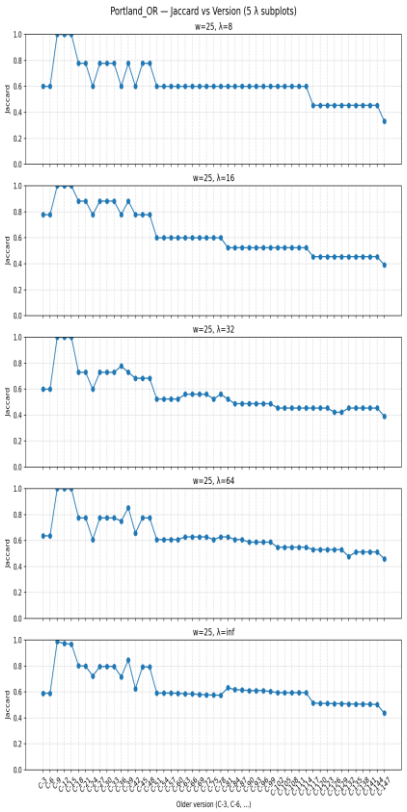

Orlando_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

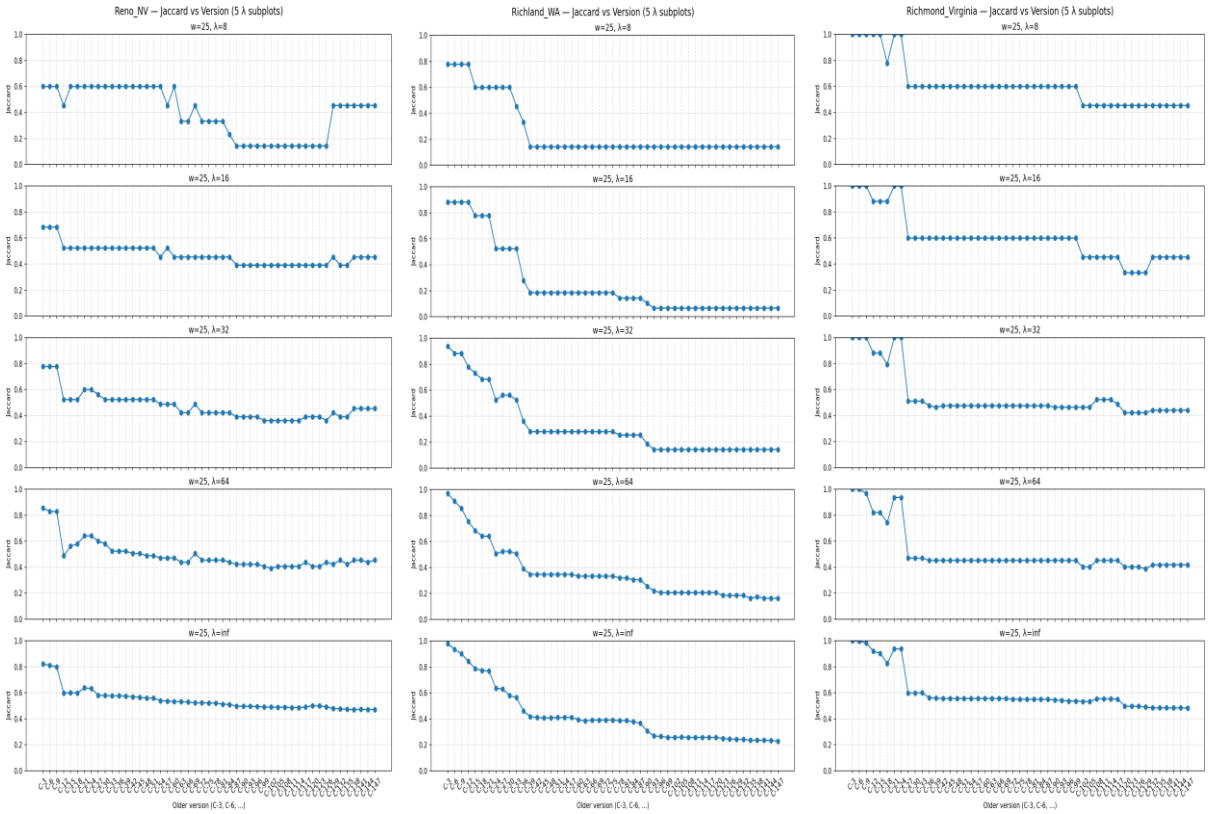
Philadelphia_PA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

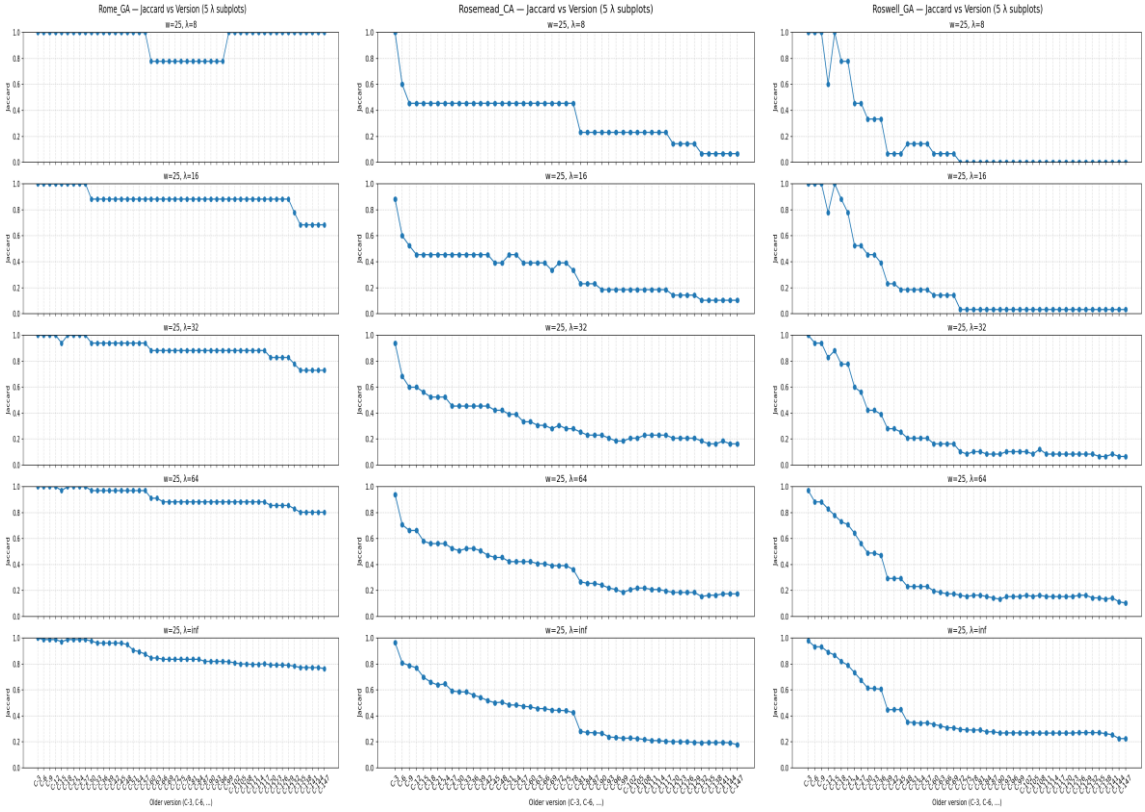
Phoenix_AZ — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

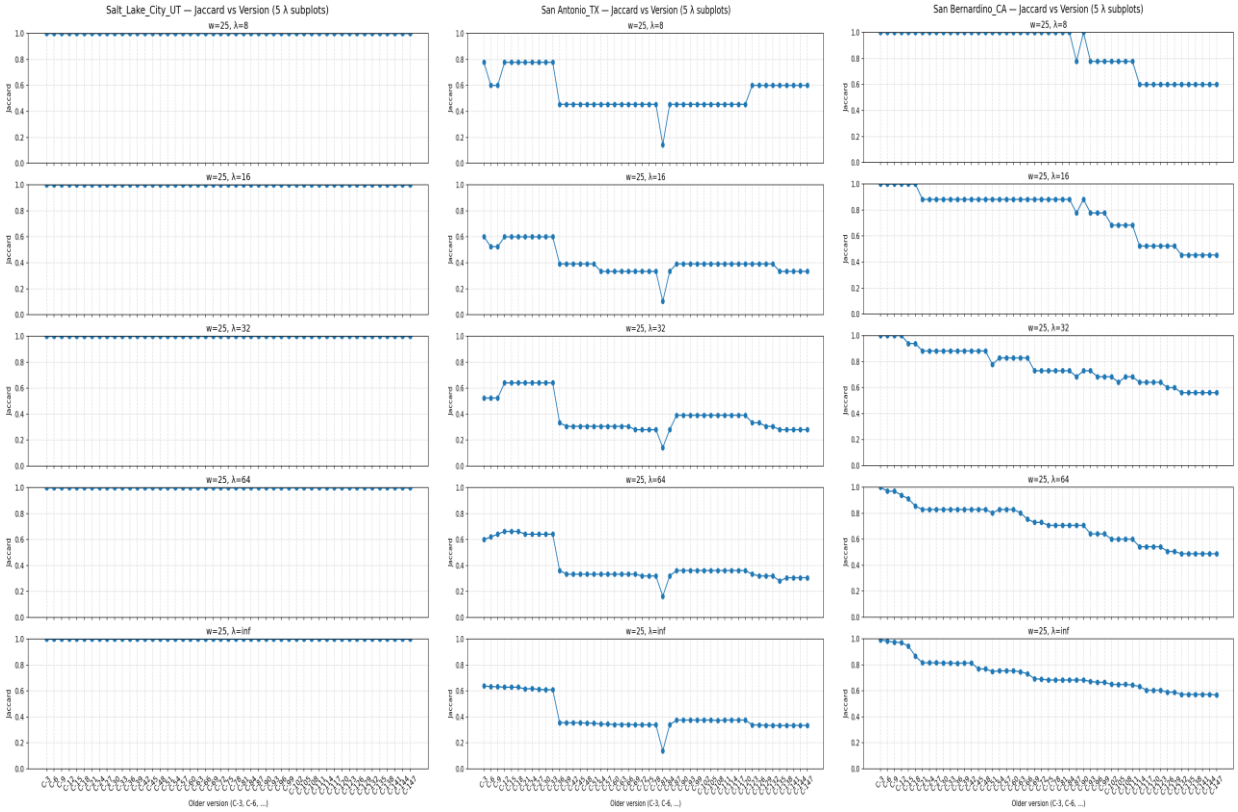

Pittsburg_PA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Pleasanton_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Portland_OR — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

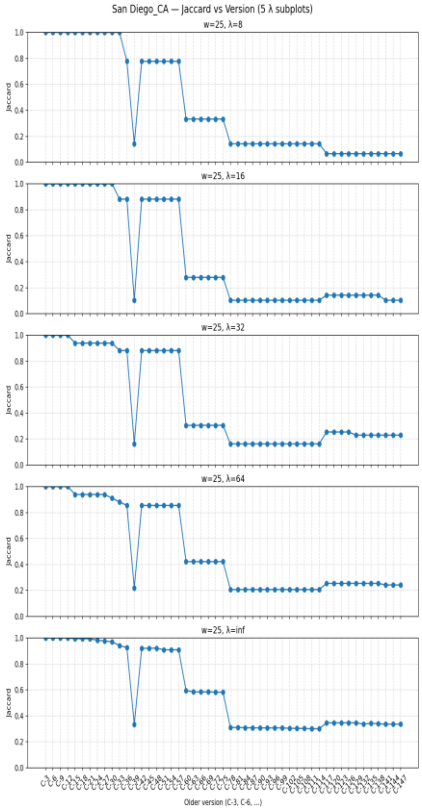

Reno_NV — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)
Richland_WA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)
Richmond_Virginia — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

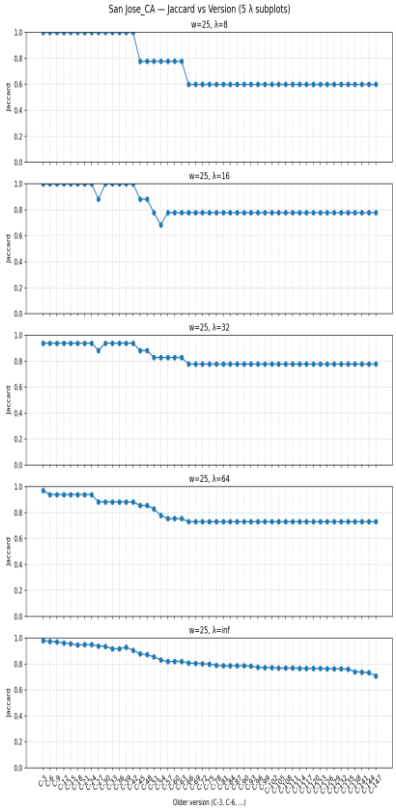

Rome_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)
Rosemead_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)
Roswell_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

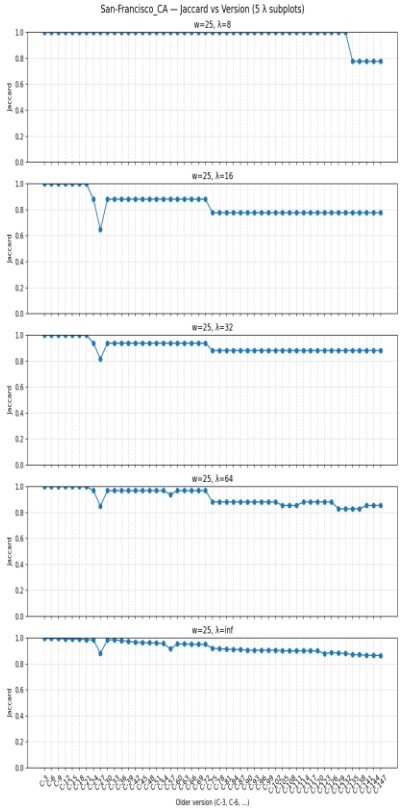

Salt_Lake_City_UT — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

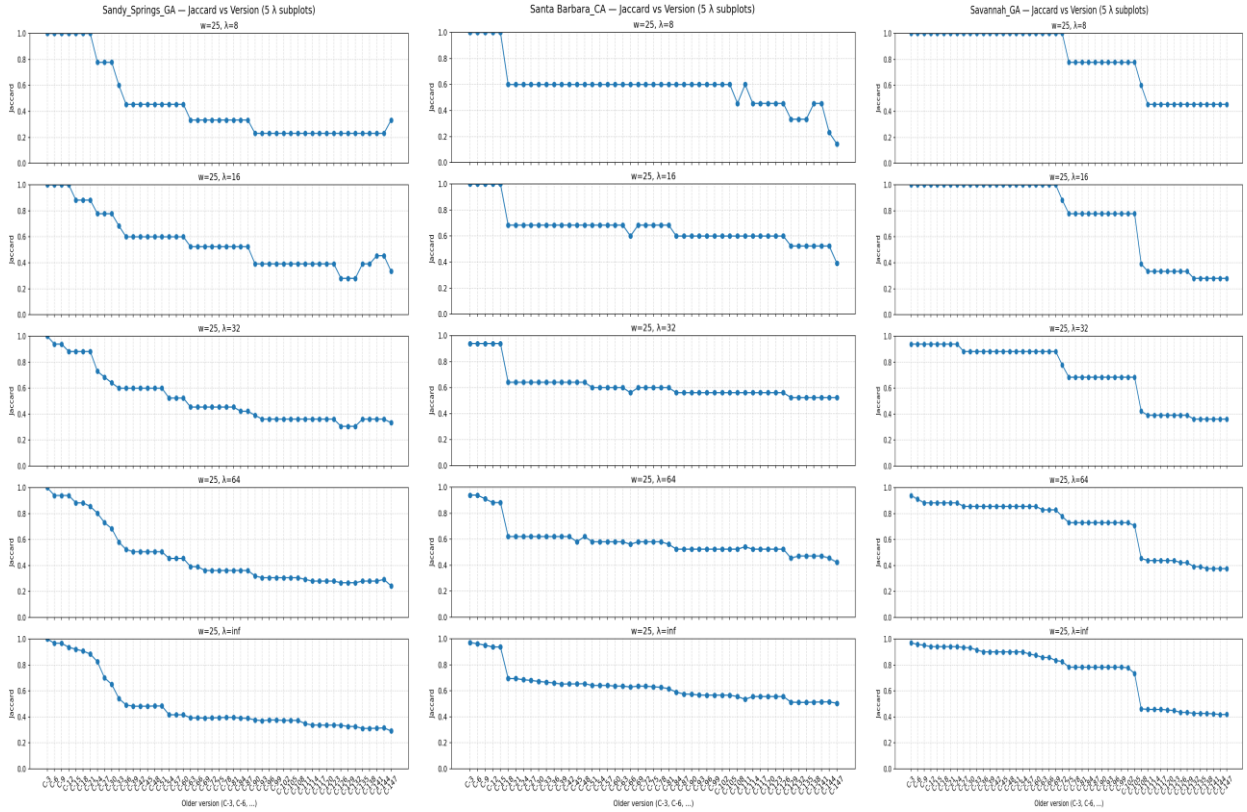
San Antonio_TX — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

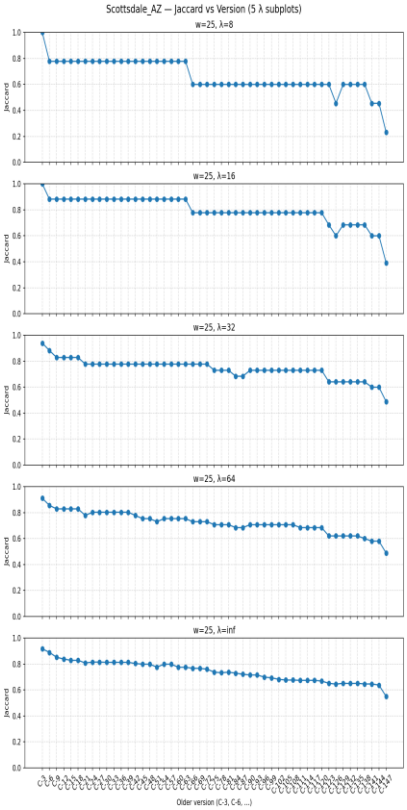
San Bernardino_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

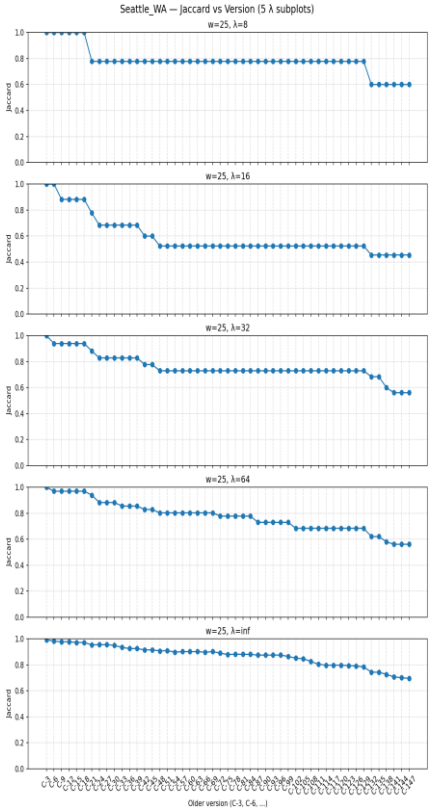

San Diego_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

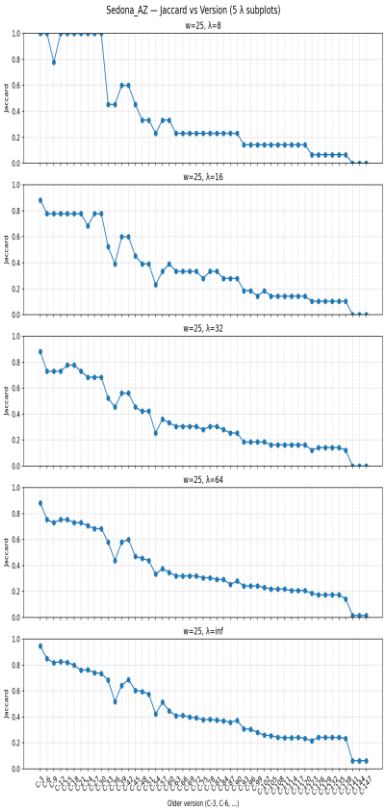

San Jose_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

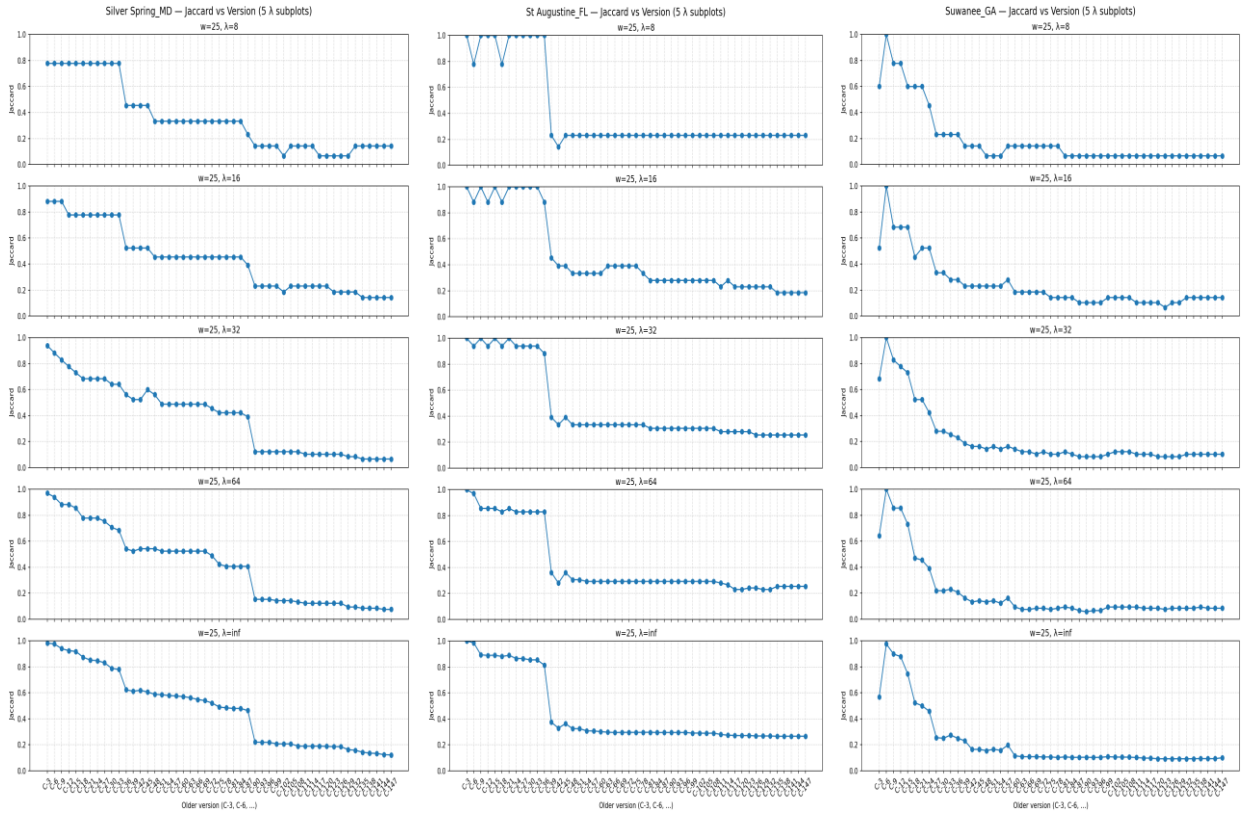

San-Francisco_CA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Sandy_Springs_GA — Jaccard vs Version (5 λ subplots)
Santa Barbara_CA — Jaccard vs Version (5 λ subplots)
Savannah_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=Inf
Jaccard
Older version (C-3, C-6, ...)


Scottsdale_AZ — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

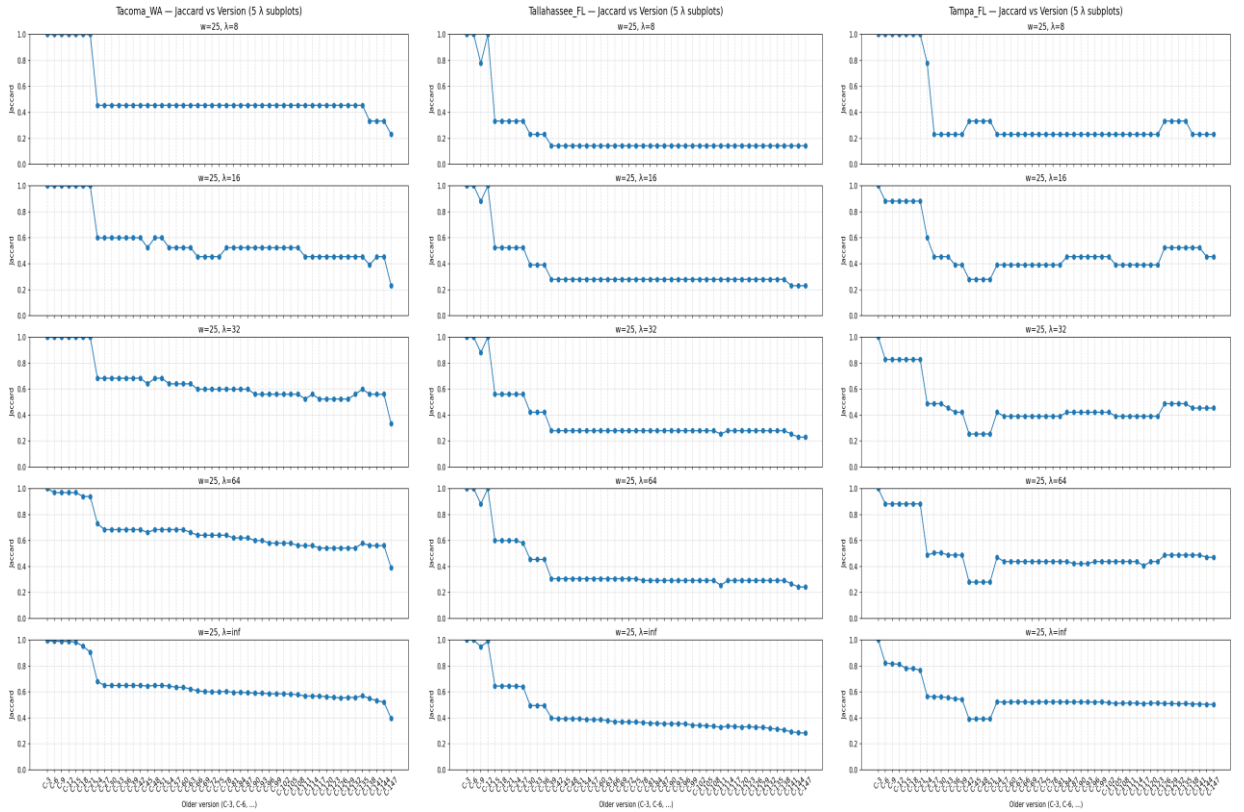

Seattle_WA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Sedona_AZ — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Silver Spring_MD — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

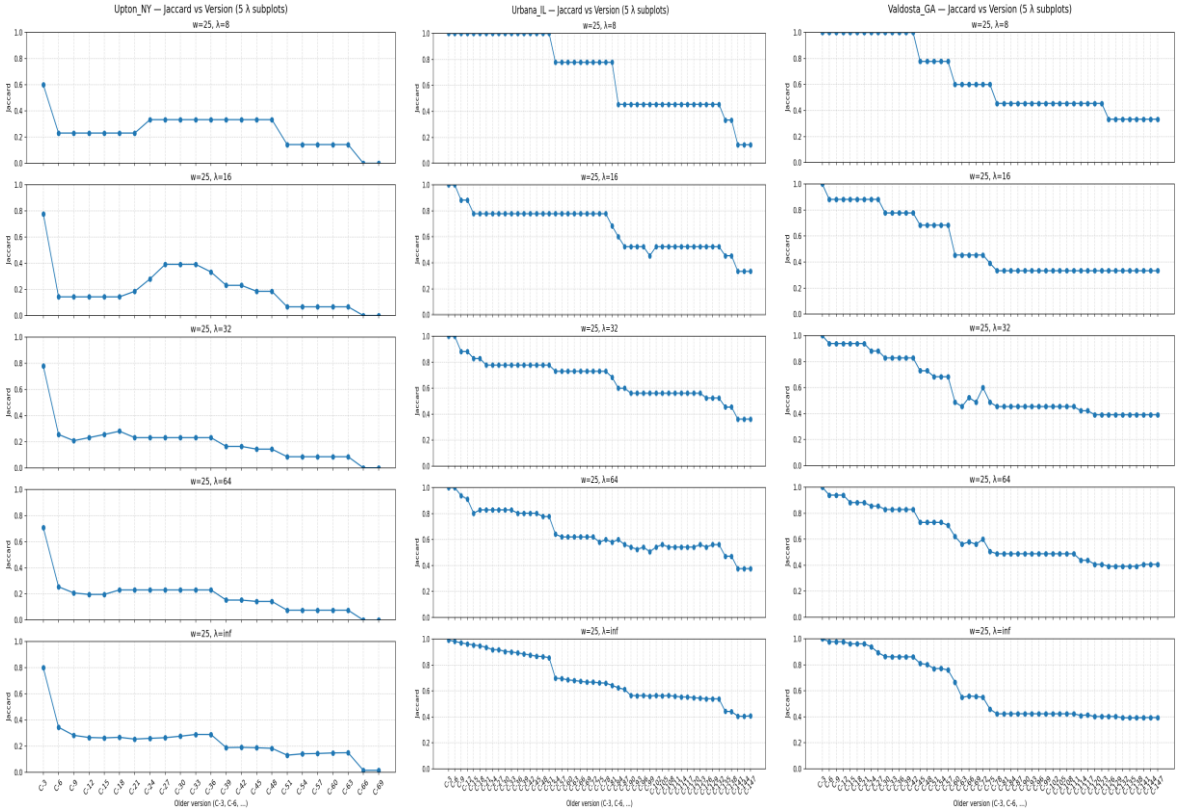
St Augustine_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

Suwanee_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Tacoma_WA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

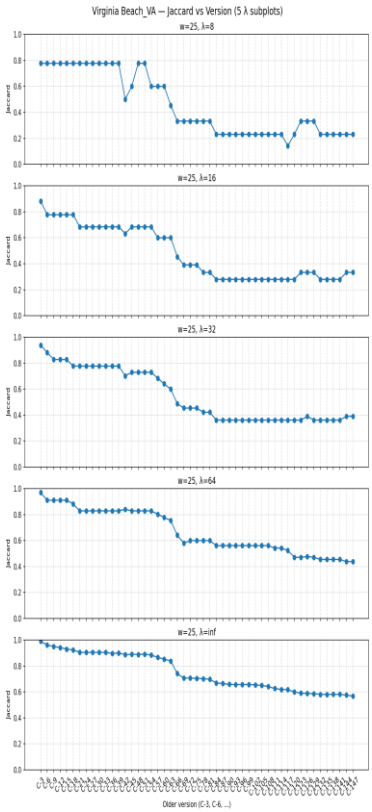
Tallahassee_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

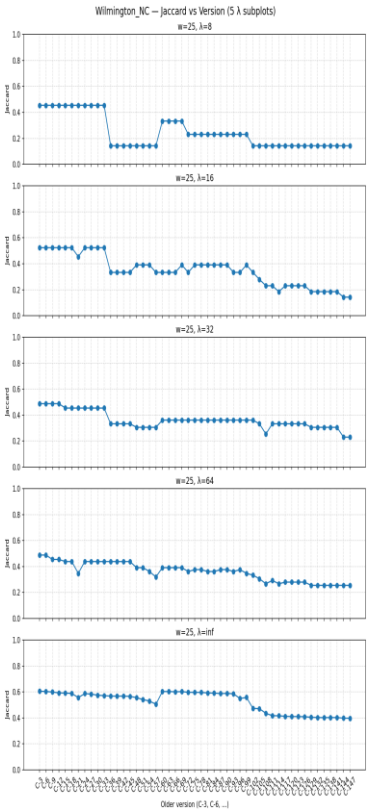
Tampa_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

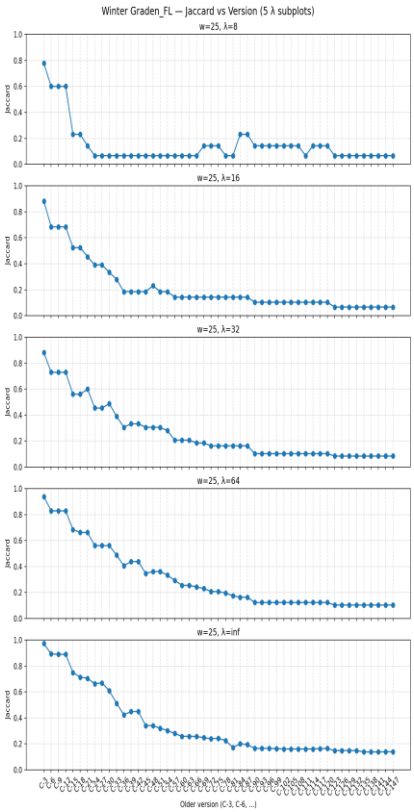

Upton_NY — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

Urbana_IL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)

Valdosta_GA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Virginia Beach_VA — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Wilmington_NC — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)


Winter Graden_FL — Jaccard vs Version (5 λ subplots)
w=25, λ=8
w=25, λ=16
w=25, λ=32
w=25, λ=64
w=25, λ=inf
Jaccard
Older version (C-3, C-6, ...)