CS768: Citation Graph Construction

Aryan Katiyar, Kunal Chaudhari, Aditya Agrawal, Navnit Thakur

April 2025

Project Repository: https://github.com/cesiumstar/CS768ASS

1 Task 1: Citation Graph Construction

We aim to build a directed citation graph from a collection of academic papers. Each node in the graph corresponds to a paper, and a directed edge from node A to B indicates that paper A cites paper B. The data is structured such that each paper is stored in a dedicated folder, containing its title and reference metadata.

1. Folder Structure and Input Data

Each folder in the dataset corresponds to one paper and contains:

- `title.txt` – contains the paper title
- `abstract.txt` – contains the paper title
- `.bib` or `.bbl` – BibTeX or LaTeX reference files

As this task focuses solely on constructing the citation graph using paper titles and their extracted citations. We do not utilize paper abstract.

2. Title Normalization and Mapping

To enable fuzzy matching of titles across references, we normalize all titles:

- Lowercase conversion
- Removal of punctuation
- Collapsing multiple spaces

```python
def normalize_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip()
```

Listing 1: Title normalization function

Using this, we build two mappings:

- `title_to_folder`: normalized title → folder name
- `folder_to_title`: folder name → normalized title

3. Citation Extraction from .bib and .bbl

We scan each folder for `.bib` and `.bbl` files and extract cited paper titles and optional years.

Example of extraction from .bib:

```
title_match = re.search(r'title\s*=\s*\{([^}]*)\}', entry)
if title_match:
    title = normalize_text(title_match.group(1))
```

Example of extraction from .bbl:

```
for line in lines:
    if '\\newblock' in line:
        title_line = line.replace('\\newblock', '').strip()
        title_parts.append(title_line)
```

We store these in a dictionary:

```
cited_titles_with_years[normalized_title] = extracted_year
```

4. Fuzzy Matching with Dataset Titles

We match each cited title to the dataset using difflib.get_close_matches() with a cutoff of 0.90 to allow slight mismatches.

```
match = get_close_matches(cited_title, title_keys, n=1, cutoff=0.90)
if match:
    matched_title = match[0]
    target_folder = title_to_folder[matched_title]
```

5. Temporal Consistency Checks

To prevent future citations, we compare the citing and cited paper's arXiv-derived year/month:

- If cited year > citing year, skip.
- If both have same year but cited month > citing month, skip.
- Self-citations (same folder) are excluded.

```
if target_year > citing_year or \
   (target_year == citing_year and target_month > citing_month):
    continue
```

6. Parallel Graph Construction

Citation extraction is parallelized using `ProcessPoolExecutor`. Each folder is processed independently.

```
with ProcessPoolExecutor() as executor:
    futures = [executor.submit(process_single_paper, folder, ...)]
```

We write:

- `edges_parallelized.txt` – citation edges
- `nodes_parallelized.txt` – paper list
- `log.txt` – logs and match details

7. Final Graph and Pickling

We use `networkx.DiGraph` to store the final citation graph. Nodes represent papers; edges represent citations.

```
G = nx.DiGraph()
G.add_edge(citing_paper, cited_paper)
pickle.dump(G, open('citation_graph.pkl', 'wb'))
```

9. Graph Analysis Results

We analyze the undirected version of the citation graph to gain structural insights and finding diameter in an undirected graph makes much more sense.

Key Statistics:

- **Number of Edges:** 30,780
- **Number of Isolated Nodes:** 401
- **Average Degree:** 9.41
- **Number of Connected Components:** 423
- **Size of Largest Component:** 6094 nodes
- **Maximum Diameter (among all components):** 13 (in the largest component)

Degree Distribution

The majority of nodes in the citation graph have a small degree. A small number of papers are highly cited (high degree). We present two views of the degree histogram.

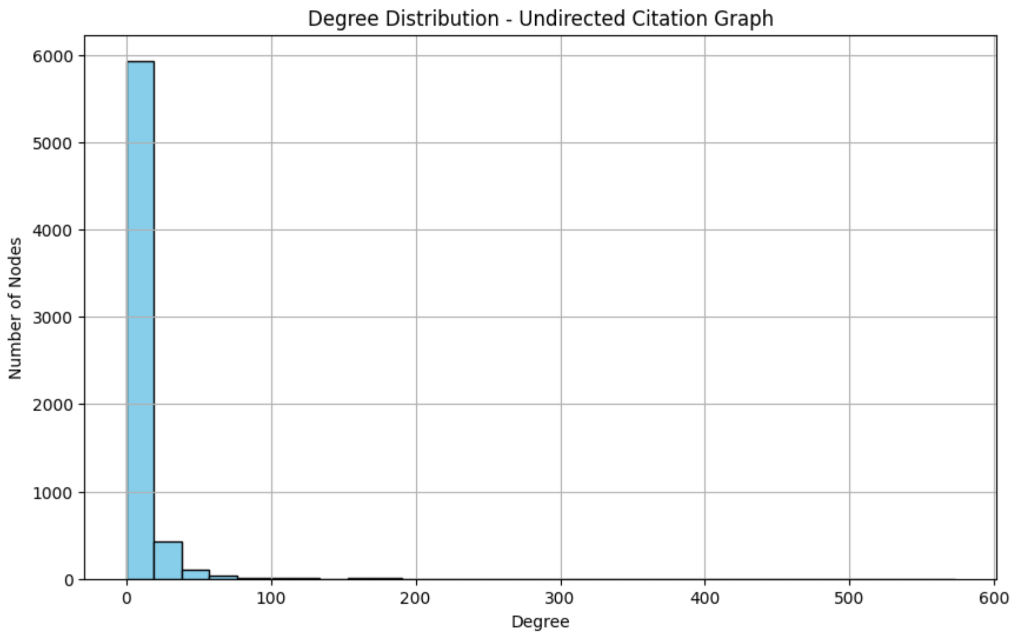

Figure 1: Degree Distribution – Undirected Citation Graph

Observations:

- A heavy-tail distribution is evident, with most nodes having low degrees.
- The presence of hubs (papers with high degree) skews the distribution.
- Over 400 nodes are completely isolated, suggesting missing or unmatched references.
- The largest connected component contains most of the citation structure and is used for diameter analysis.

Diameter Analysis

We compute the diameter for each connected component in the undirected graph. The largest diameter of 13 occurs in the largest component of size 6094. Smaller components, typically of size 2–4, have diameters in the range 1–3.

Code Used for Graph Analysis:

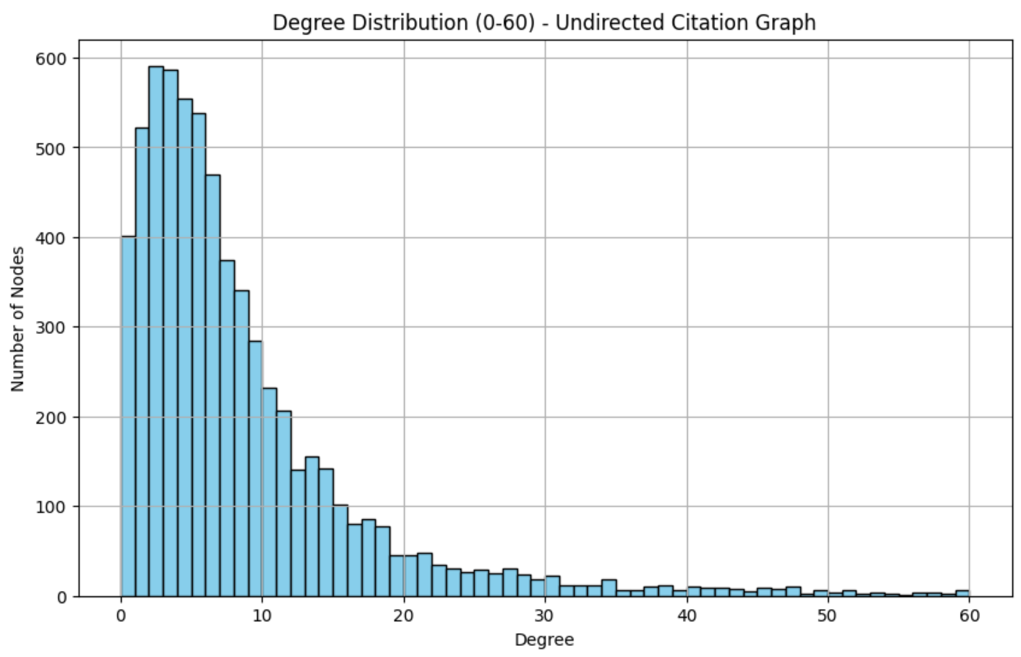

Figure 2: Zoomed-in Degree Distribution (0–60) – Undirected Citation Graph

```python
G_undirected = G.to_undirected()
components = list(nx.connected_components(G_undirected))
diameter = nx.diameter(G_undirected.subgraph(components[0]))
```

This analysis provides a global picture of the citation topology and highlights the importance of large connected cores for citation-based tasks.

2 Task 2: Citation Link Prediction in Academic Graphs

In this task, we want to highlight two approaches that we leveraged and the intuition behind it.

2.1 Approaches

We predict citation links using two approaches:

(a) A classifier-based link predictor.

(b) A similarity-weighted citation voting model.

2.2 Approach (a): Classifier-Based Link Prediction

We define each training sample as a directed pair (u, v):

- Positive pairs: where u actually cites v.
- Negative pairs: randomly sampled node pairs (u, v) without a citation

Features: In-neighbor similarity statistics with v:

- Average of top 5 similarities of u with the *citers* of v.
- Average of top 10 similarities
- Maximum of the similarities
- Number of citers of v

That's it. We found out the node embeddings of the using *sentence-transformer*, and the similarity using cosine similarity. Since the features are very general, overfitting is almost impossible.

2.2.1 Model

We basically want to approximate the function f such that f(features of pair (x,y)) = 1 if there is an edge else is 0. Now the challenge is that if the negative samples are too large, the function approximation will tilt towards predicting a non-edge. Further, a Random Forest classifier is trained and validated on disjoint node sets (no node u appears in both train and validation for fairness).

2.2.2 Design Intuition and Reasoning

- **Why model $f(u, v)$ as a classification problem?**
  Citation is a directed binary relation: a paper either cites another or does not. This naturally lends itself to a binary classification formulation,
- **Why not use raw embedding similarity between u and v?**
  Direct cosine similarity between u and v may not capture the local context of citations. A paper u often cites v not because they are semantically identical, but because u aligns with a *community* of papers that cite v. Therefore, modeling how similar u is to the in-neighbors of v provides a more informative signal.
- **Why minimal use of Graph Structure?**
  . We haven't really benefited from the structure of the Graph except the immediate neighbors. It is very easy to make the model complicated, miss the primary signals and overfit on the limited number of papers. Since correlation is high enough with the neighbors, we have kept the model simple and practical. The use of GNN seemed too complicated for our tiny brains to handle tbh.
- **Why include multiple statistics (mean, max, count)?**
  Different summary statistics of similarity capture different citation signals:
  - *Mean similarity:* overall topical alignment between u and the citation community of v.
  - *Max similarity:* checks for strong one-to-one analogies.
  - *Count of in-neighbors:* a proxy for v's citation popularity or contextual breadth.

  In simple words, a paper is likely to be cited if it is popular or if similar papers cite it. These features prompt model to find a balance between the two.
- **Why split the dataset based on source nodes u only?**
  To simulate the task of predicting citations for newly introduced papers, we split the dataset such that validation contains unseen source nodes u. This ensures a fair evaluation and prevents data leakage from citation patterns observed in training.
- **Why use Random Forest instead of deep models?**
  Our feature set is small and interpretable (4 features per (u, v) pair).

2.3 Approach (b): Similarity-Based Citation Aggregation

This non-parametric approach ranks citations by:

1. Identifying top-m similar papers to a given query node.
2. Aggregating citations of those similar papers, weighted by exp(`similarity`).

The top-k scored papers are predicted as likely citations. The following code snippet almost explains the crux of the approach

```
    citation_scores = defaultdict(float)
    for sim_score, similar_node in top_m:
        weight = np.exp(sim_score)
        for cited in G.successors(similar_node):
            citation_scores[cited] += weight
```

2.4 Evaluation Protocol

We perform evaluation on 50 randomly sampled nodes using the following steps:

- Predict top-k citations using both methods.
- Compute **Recall@k**: proportion of actual (in-graph) citations present in top-k predictions.

2.5 Results and Analysis

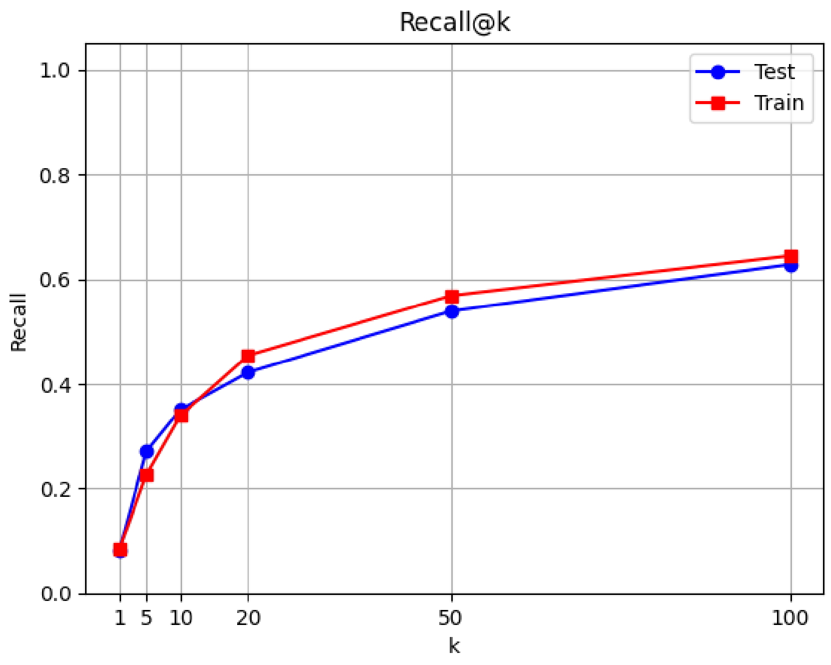

Figure 3: Recall@k averaged over 50 random papers in validation (ML based approach)

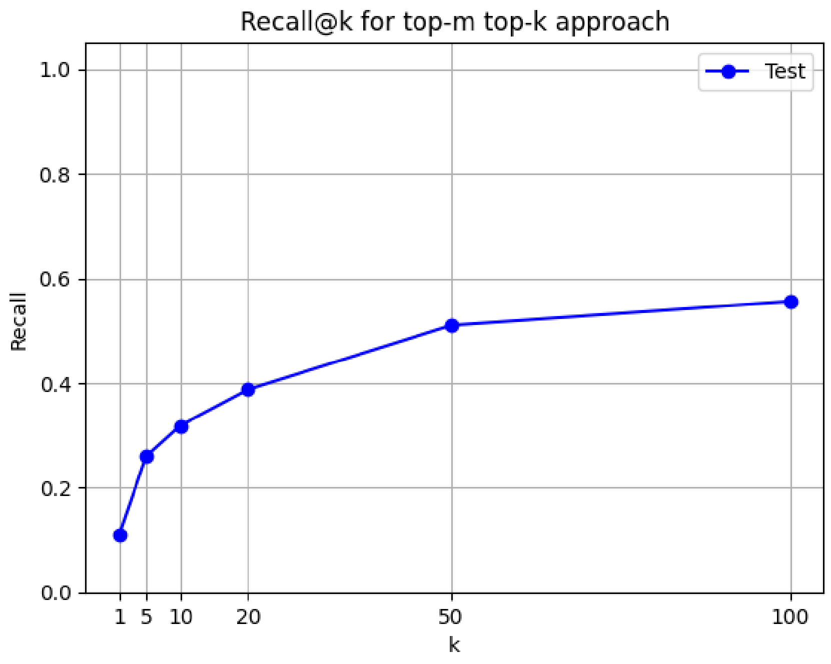

Figure 4: Recall@k averaged over 50 random papers in validation (voting based approach)

The second approach (non ML based) actually works better than model based for low k. This does point to how Model is actually bad in capturing when the features do not have much to present, or there is to actually capture other than the simplest of features. For larger k, since top m similar papers has m fixed, it plateaus for larger k values. The papers which have been cited many times polarize the voting based approach where it gets a large score. A good approach can be to combine these models to form q hybrid, or capture the features of the first approach in the second one, but we were not able to do in the prescribed time.

Another noteworthy fact is that train set has almost the same recall. This is due to the fact that features do not encompass exact information of a given node at all, and due it's simplistic nature it generalizes over all nodes. So it is very *improbable* for train data to perform considerably better than the validation set.