Lab 1 Big Data Management
Problem A

Alejandro Delgado, Viktoria Gagua

April 29, 2025

Task A.1 (Modeling)

1. Visual Representation

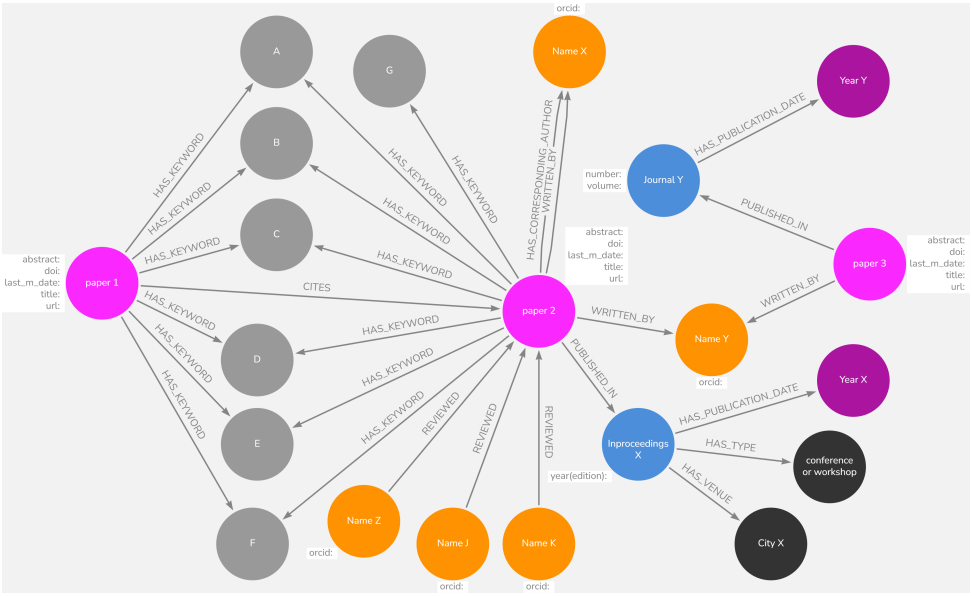

Figure 1: Visual Representation of the Graph

2. Justification of design

Maintenance and Reusability

Our proposed model has clear separation of entities, normalized structure, and is flexible for extensions. By modeling distinct elements as separate node types, we follow separation of concerns principles, making maintenance easier as changes to one entity do not affect others. The normalized approach avoids data duplication for recurring entities, ensuring consistency and reducing redundancy. This design can be easily extended to include additional types or relationships without restructuring the entire graph, supporting future requirements.

Query Performance Optimization

Our model optimizes query performance through direct relationship traversal, enabling efficient one-hop queries like finding all papers by an author. Strategic relationship design, such as HAS CORRESPONDING AUTHOR, streamlines identifying primary contributors. The separate Time nodes connected to publications enhance temporal query efficiency. Topic indexing through the HAS KEYWORD relationship facilitates content-based searches, while the bidirectional citation structure supports analysis of citation counts and impact factors.

Design Choices for Specific Requirements

Our model addresses specific requirements through strategic design choices. The Type nodes distinguish between conferences and workshops, enabling event type filtering. Separately stored venue information supports geographical analysis and location queries. The hierarchical relationship between papers and publication venues (journals or proceedings) allows for venue-based paper grouping.

Task A.2 (Instantiation/Loading)

How the Data Was Generated

Base Data Source: The DBLP XML dataset was converted to CSV format using the XMLToCSV.py tool from GitHub (ref: *https://github.com/ThomHurks/dblp-to-csv*). Data cleaning was performed to remove empty rows, handle bad quotations, and delete non-paper entries such as "Preface." For computational efficiency, the dataset was limited to 2,000 articles and 2,000 inproceedings entries. Since the DBLP dataset lacks some required information, synthetic data was added, including randomly assigned venues from a predefined list of cities, random classification of each inproceedings as either "conference" or "workshop," realistic paper summaries generated using the Faker library, and randomly assigned keywords from a curated list of 50 computer science topics. Reviewers were selected from authors who published at least two papers, ensuring no author reviews their own paper. Citations were created based on keyword similarity, where papers sharing at least six keywords are connected through citation relationships.

Assumptions in our Model

The first author listed for each paper is assumed to be the corresponding author. Publication date is tracked by year, with more granular timing represented by the last_m_date attribute representing the last modification date (paper updates). Each journal is uniquely identified by its name and volume, with the number attribute capturing issue information. Events (conferences/workshops) are uniquely identified by their booktitle (which corresponds to the event name) and year combination, with papers published at the same event sharing identical venue and type. Each paper has exactly three reviewers, all selected from authors with at least two publications. Citations are directional, with later papers citing earlier ones based on publication dates. Each paper is assigned five to ten keywords representing its main topics. Papers are uniquely identified by their titles, and we ensured that no duplicate titles exist in the dataset (in the data pre-processing).

Task A.3 (Evolving the graph)

For storing review data, we thought that creating a dedicated `Review` node type is more optimal than enhancing the `REVIEWED` relationship with properties, especially for complex

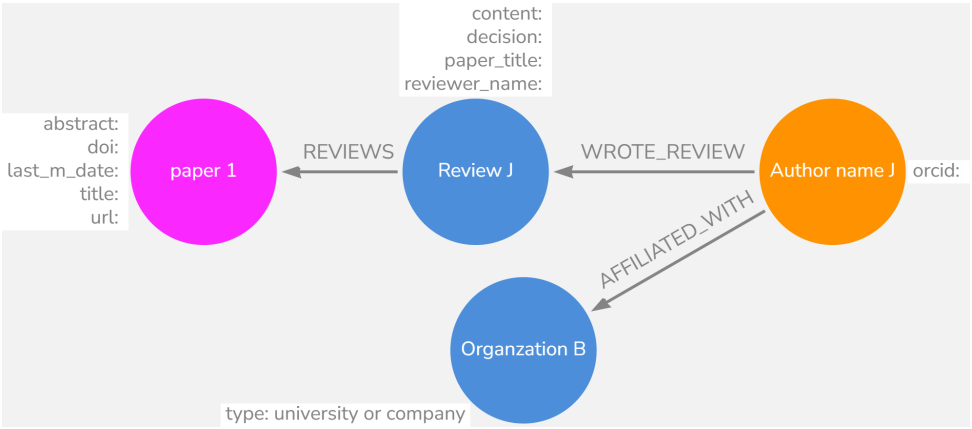

Figure 2: Visual Representation of the Graph Modifications (Reviews)

querying. Neo4j performs node property querying more efficiently than filtering relationship properties.

Our proposed modification of the graph model includes adding **2 new nodes**: `Review` (with properties for content and accept/reject decision) and `Organization` (with name and type properties). These connect through new relationships: `WROTE_REVIEW` (Author to Review), `REVIEWS` (Review to Paper), and `AFFILIATED_WITH` (Author to Organization). The Review node separates review data clearly, improves query performance for counting decisions, ensures one decision per review, and enables detailed reviewer analytics. This dedicated node structure easily accommodates future extensibility such as review dates, revision rounds, or detailed scoring. Review nodes also simplify and accelerate decision counting for acceptance calculations and reviewer behavior analysis. The Organization node normalizes shared affiliations, supports institution-specific paper queries, and efficiently stores additional organizational attributes.

Appendix

Listing 1: Query for problem set A

```
//  DBLP GRAPH INGEST - Problem A
//
//  SECTION A.2    JOURNAL PAPERS (dblp_article_clean.csv)
//

// We add a uniqueness constraint on (name, volume) so we
//    never end up with duplicate Journal nodes when the script
//    is  rerun .  "IF NOT EXISTS" keeps it idempotent.
CREATE CONSTRAINT journal_unique IF NOT EXISTS
FOR (j:Journal)
REQUIRE (j.name, j.volume) IS UNIQUE;

// We read the CSV, telling Neo4j that fields are separated by
//    semicolons. Rows with missing or blank titles are ignored
//    because a paper without a title isnt helpful to us.
LOAD CSV WITH HEADERS FROM 'file:///dblp_article_clean.csv' AS row
   FIELDTERMINATOR ';'
WITH row WHERE row.title IS NOT NULL AND trim(row.title) <> ''

// We create (or reuse) a Paper node per *title*. Extra metadata
//    (mdate, DOI, URL, abstract) is stored so later queries dont
//    need to go back to the raw files.
MERGE (p:Paper {title: row.title})
SET p.last_mdate = row.mdate,
   p.doi       = row.ee,
   p.url       = row.url,
   p.abstract  = row.abstract

// For each (journal, volume) pair we merge a Journal node and
//    keep the latest issue number weve seen. We decided to do
//    that because many journals reuse the same volume number for
//    multiple issues and we only care about the most recent one.
MERGE (j:Journal {name: row.journal, volume: row.volume})
SET j.number = row.number

// We connect every Paper to its Journal via [:PUBLISHED_IN].
MERGE (p)-[:PUBLISHED_IN]->(j)

// We attach each Journal to a Time node (year granularity).
//    That way, timeseries queries run fast and stay clean.
MERGE (t:Time {year: row.year})
MERGE (j)-[:HAS_PUBLICATION_DATE]->(t)
```

```
// Author handling   we split the author string on "|" so we can
//     iterate over individual names. We keep ORCIDs when theyre
//     present, otherwise we leave the property null.
WITH row, p,
     split(row.author, '|')      AS authors,
     split(row.`author-orcid`, '|') AS orcids
UNWIND range(0, size(authors)-1) AS idx
WITH row, p,
     authors[idx]                                                    AS author_name
         ,
     CASE WHEN idx < size(orcids) THEN orcids[idx] ELSE NULL END  AS
         author_orcid,
     idx
MERGE (a:Author {name: author_name})
SET a.orcid = author_orcid
MERGE (p)-[:WRITTEN_BY]->(a)

// We treat the first author in the list as the corresponding
//     authorsimple  heuristic but good enough for demo purposes.
WITH row, p, a, idx WHERE idx = 0
MERGE (p)-[:HAS_CORRESPONDING_AUTHOR]->(a)

// Keyword handling   we decode the JSON list with APOC, then we
//     connect the Paper to Topic nodes (creating them on demand).
WITH row, p
WITH p, apoc.convert.fromJsonList(row.keywords) AS keyword_list
UNWIND keyword_list AS kw
MERGE (t:Topic {name: kw})
MERGE (p)-[:HAS_KEYWORD]->(t);

//


//  SECTION A.2 CONFERENCE PAPERS (dblp_inproceedings_clean.csv)
//



// We set up a uniqueness constraint so each (booktitle, year)
//     pair gives us exactly one Inproceedings node.
CREATE CONSTRAINT inproceedings_unique IF NOT EXISTS
FOR (i:Inproceedings)
REQUIRE (i.booktitle, i.year) IS UNIQUE;
```

```
// Same CSVloading pattern, we skip blank titles again.
LOAD CSV WITH HEADERS FROM 'file:///dblp_inproceedings_clean.csv' AS row
    FIELDTERMINATOR ';'
WITH row WHERE row.title IS NOT NULL AND trim(row.title) <> ''

// Paper node.
MERGE (p:Paper {title: row.title})
SET p.last_mdate = row.mdate,
    p.doi       = row.ee,
    p.url       = row.url,
    p.abstract  = row.abstract

// Inproceedings node.
MERGE (i:Inproceedings {booktitle: row.booktitle, year: row.year})

// Paper    Inproceedings link.
MERGE (p)-[:PUBLISHED_IN]->(i)

// We wire the conference instance to its Time node (year).
MERGE (t:Time {year: row.year})
MERGE (i)-[:HAS_PUBLICATION_DATE]->(t)

// We decided to store the submission type (full/short/etc.)
//     as its own node so we can quickly group papers later.
MERGE (type:Type {name: row.type})
MERGE (i)-[:HAS_TYPE]->(type)

// Venue node lets us query papers by conference location or
//     series host without stringmatching.
MERGE (v:Venue {name: row.venue})
MERGE (i)-[:HAS_VENUE]->(v)

// Author parsing   identical to the journal branch.
WITH row, p,
     split(row.author, '|')      AS authors,
     split(row.`author-orcid`, '|') AS orcids
UNWIND range(0, size(authors)-1) AS idx
WITH row, p,
     authors[idx]                                                AS author_name
         ,
     CASE WHEN idx < size(orcids) THEN orcids[idx] ELSE NULL END AS
         author_orcid,
     idx
MERGE (a:Author {name: author_name})
SET a.orcid = author_orcid
```

```
MERGE (p)-[:WRITTEN_BY]->(a)

// First author   corresponding.
WITH row, p, a, idx WHERE idx = 0
MERGE (p)-[:HAS_CORRESPONDING_AUTHOR]->(a)

// Keywords.
WITH row, p
WITH p, apoc.convert.fromJsonList(row.keywords) AS keyword_list
UNWIND keyword_list AS kw
MERGE (t:Topic {name: kw})
MERGE (p)-[:HAS_KEYWORD]->(t);

//


//  SECTION    REVIEWS & CITATIONS
//



// We import reviewerpaper pairs and attach them directly with
//    [:REVIEWED]. This gives us a foothold to build richer review
//    entities later.
LOAD CSV WITH HEADERS FROM 'file:///review_edges.csv' AS row FIELDTERMINATOR ';
    '
WITH row WHERE row.reviewer IS NOT NULL AND row.paper IS NOT NULL
MATCH (a:Author {name: row.reviewer})
MATCH (p:Paper {title: row.paper})
MERGE (a)-[:REVIEWED]->(p);

// Citations   our heuristic: if two papers share at least
//    six keywords, the earlier one cites the later one. We add
//    an extra  idbased  tiebreaker  so the direction is deterministic.
MATCH (p1:Paper)-[:HAS_KEYWORD]->(k:Topic)<-[:HAS_KEYWORD]-(p2:Paper)
WHERE p1 <> p2
WITH p1, p2, COUNT(DISTINCT k) AS shared_keywords
WHERE shared_keywords >= 6
  AND ( p1.last_mdate < p2.last_mdate
       OR (p1.last_mdate = p2.last_mdate AND id(p1) < id(p2)) )
MERGE (p1)-[:CITES]->(p2);

//
```

```
// SECTION A.3   SYNTHETIC AFFILIATIONS & FULL REVIEWS
//



// We set up constraints for Organization and Review nodes so we
//     cant  accidentally duplicate them.
CREATE CONSTRAINT organization_name_unique IF NOT EXISTS
FOR (o:Organization) REQUIRE o.name IS UNIQUE;

CREATE CONSTRAINT review_unique IF NOT EXISTS
FOR (r:Review) REQUIRE (r.reviewer_name, r.paper_title) IS UNIQUE;

// We create a small list of universities and companies, then we
//    MERGE a node for each one with a type label. Having both
//    academia and industry lets us demo queries on collaboration.
WITH [
  'Stanford University', 'MIT', 'Berkeley', 'Harvard', 'Oxford',
  'Cambridge', 'ETH Zurich', 'Technical University of Munich',
  'University of Tokyo', 'Tsinghua University'
] AS universities,
[
  'Google', 'Microsoft', 'Apple', 'Meta', 'Amazon',
  'IBM', 'Intel', 'Nvidia', 'Baidu', 'Samsung'
] AS companies
UNWIND universities AS uni_name
MERGE (o:Organization {name: uni_name, type: 'university'})

WITH companies
UNWIND companies AS comp_name
MERGE (o:Organization {name: comp_name, type: 'company'});

// For demo simplicity we randomly assign exactly one affiliation
//    to every author. In production wed of course use real data.
MATCH (a:Author)
WITH a, rand() AS r
MATCH (o:Organization)
WITH a, o, r ORDER BY r LIMIT 1
MERGE (a)-[:AFFILIATED_WITH]->(o);

// Now we replace the simple [:REVIEWED] edge with a full Review
//    node that stores text + accept/reject. We chose a 70/30 accept
//    ratio and some templated prose so we can explore review text in
//    NLP demos.
MATCH (a:Author)-[r:REVIEWED]->(p:Paper)
```

```
WITH a, p
WITH a, p,
  CASE WHEN rand() > 0.3 THEN 'accept' ELSE 'reject' END AS decision,
  'This paper ' +
  CASE WHEN rand() > 0.5 THEN 'presents an interesting approach to ' ELSE '
     proposes a novel method for ' END +
  'the problem. ' +
  CASE WHEN rand() > 0.3 THEN 'The methodology is sound and the results are
     convincing. ' ELSE 'The experimental evaluation could be more thorough. '
      END +
  CASE WHEN rand() > 0.7 THEN 'Overall, a strong contribution to the field.'
     ELSE 'Some claims require better justification.' END AS content
MERGE (rev:Review { reviewer_name: a.name, paper_title: p.title })
SET rev.content = content,
    rev.decision = decision
MERGE (a)-[:WROTE_REVIEW]->(rev)
MERGE (rev)-[:REVIEWS]->(p)
WITH a, p
MATCH (a)-[old:REVIEWED]->(p)
DELETE old; // we dont need the placeholder edge anymore

// We flag each Paper as accepted or rejected based on majority
//    vote across its Review nodes. >=50% "accept" means accepted.
MATCH (p:Paper)<-[:REVIEWS]-(rev:Review)
WITH p, COLLECT(rev.decision) AS decisions
WITH p,
     SIZE([d IN decisions WHERE d = 'accept']) AS accept_count,
     SIZE(decisions) AS total_count
WHERE total_count > 0
SET p.accepted = CASE WHEN 1.0 * accept_count / total_count >= 0.5 THEN true
    ELSE false END;

// Finally we create three singleproperty indexes so lookups on
//    review decision, organization type, and acceptance status stay
//    snappy as the dataset grows.
CREATE INDEX review_decision_idx IF NOT EXISTS FOR (rev:Review) ON (rev.
    decision);
CREATE INDEX organization_type_idx IF NOT EXISTS FOR (o:Organization) ON (o.
    type);
CREATE INDEX paper_accepted_idx  IF NOT EXISTS FOR (p:Paper)    ON (p.accepted);

//                                                                FINALE
```