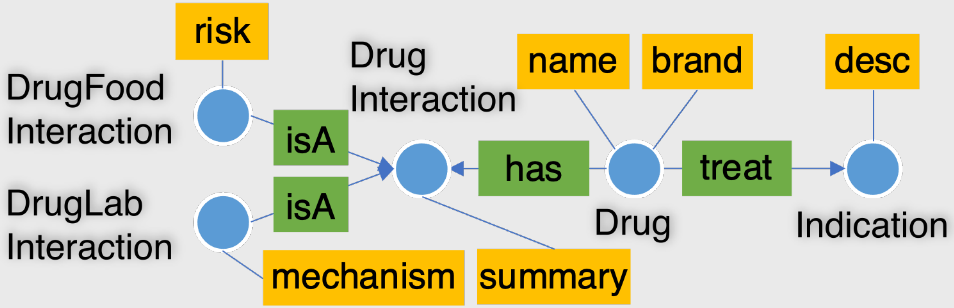
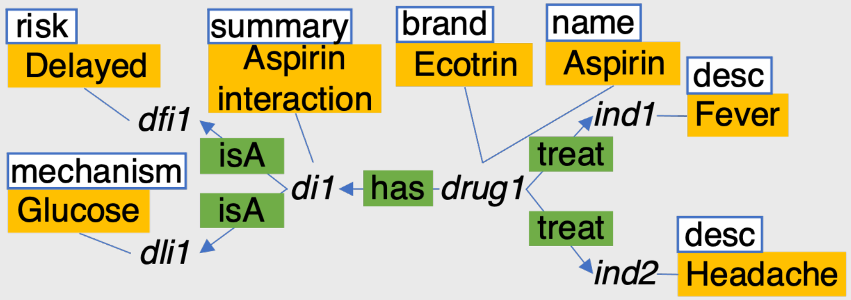
Property Graph Schema Optimization for Domain-Specific Knowledge Graphs

Rana Alotaibi	UC San Diego
Chuan Lei	IBM Research – Almaden
Abdul Quamar	IBM Research – Almaden
Vasilis Efthymiou	FORTH – ICS
Fatma Özcan	Google

Motivation

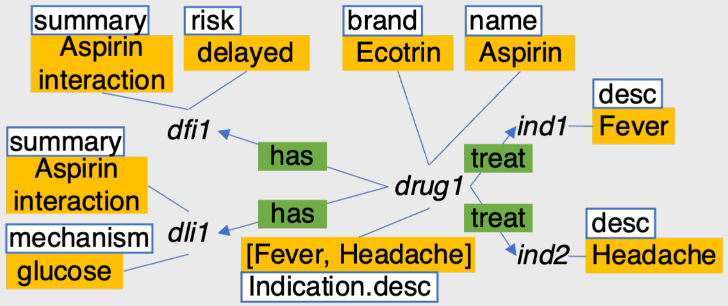


Ontology



Property Graph 1

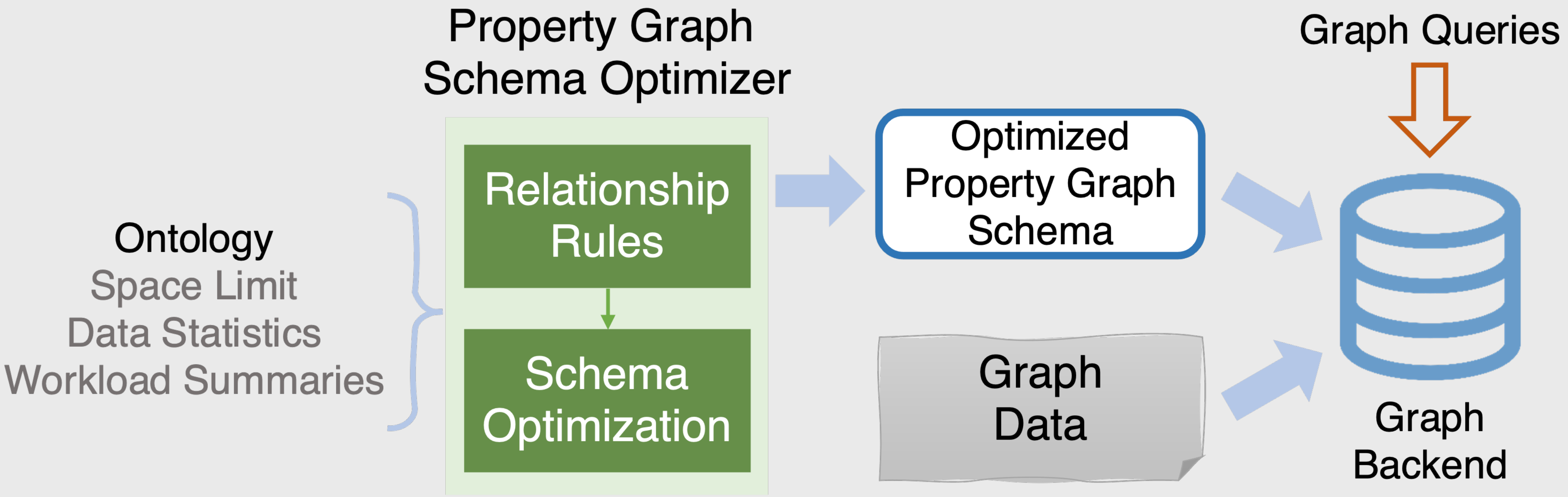
Graph pattern matching, aggregation, lookup queries



Property Graph 2

- Graph query performance varies vastly for different property graphs corresponding to different schemas
- Ontology provides unique opportunities for schema optimization

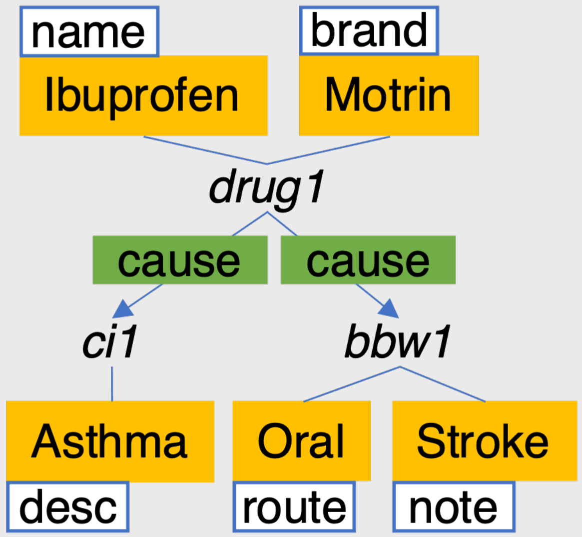
System overview



Union rule

- Union relationship
  - Union concept and member concept
  - Each instance of a union concept is an instance of one of its member concepts, and vice versa
- Directly connect the member concept to the other concepts that connect to the union concept
  - Avoid edge traversals between union and member concepts

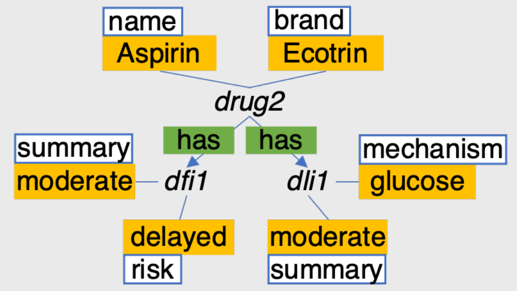
Drug (name STRING, brand STRING),
ContraIndication (desc STRING),
BlackBoxWarning (note STRING, route STRING),
(Drug)-[cause]->(ContraIndication),
(Drug)-[cause]->(BlackBoxWarning)



Inheritance rule

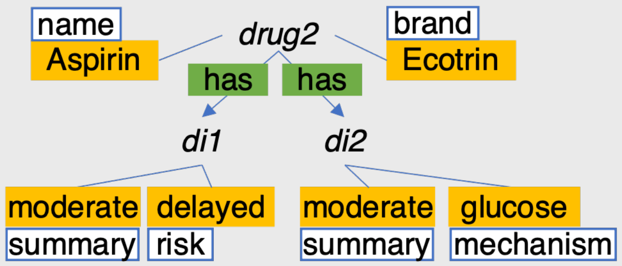
- Inheritance relationship
  - Parent and child concepts
  - Similar to union relationship, except a parent concept may have instances that are not present in any of its children
- Three scenarios
  - Scenario 1 – connect the child to the concepts associated with its parent, and attach all data properties of the parent to the child
  - Scenario 2 – connect the parent to the concepts associated with its child, and attach all data properties of the child to the parent
  - Scenario 3 – connect the parent and child with an edge of type `isA`
- Use Jaccard similarity (parent & child) to choose from three scenarios
  - Avoid edge traversals between parent and child concepts

Drug (name STRING, brand STRING),
DrugFoodInteraction (risk STRING, summary STRING),
DrugLabInteraction (mechanism STRING, summary STRING),
(Drug)-[has]->(DrugFoodInteraction),
(Drug)-[has]->(DrugLabInteraction)



Scenario 1

Drug (name STRING, brand STRING),
DrugInteraction (summary STRING, risk STRING, mechanism STRING),
(Drug)-[has]->(DrugInteraction)

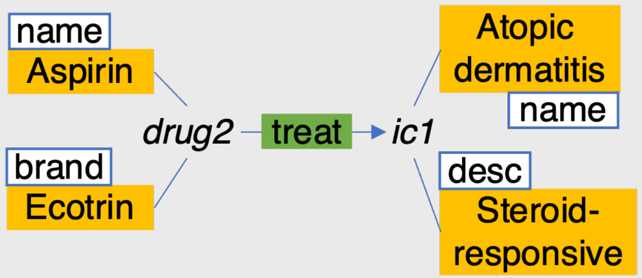


Scenario 2

One-to-one relationship rule

- One-to-one relationship
  - An instance of one concept can only relate to one instance of the other concept, and vice versa
- Represent two concepts as one combined node in the optimized schema
  - Similar to joining two tables in relational databases (one row in one table is linked with only one row in another table and vice versa)
  - Avoid edge traversals and reduce number of instances (vertices)

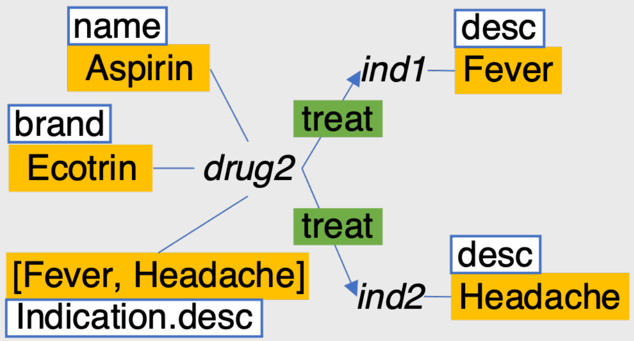
Drug (name STRING, brand STRING),
IndicationCondition (desc STRING, name STRING),
(Drug)-[treat]->(IndicationCondition)



One-to-many & many-to-many relationship rules

- One-to-many relationship
  - An instance of one concept (c_i) can potentially refer to several instances of the other concept (c_j), but not vice versa
- Many-to-many relationship
  - Equivalent to two one-to-many relationships
  - An instance of one concept (c_i) can potentially refer to several instances of the other concept (c_j), and vice versa
- Propagate each data property of c_i as a property of type `LIST` to c_j
  - Similar to the denormalization technique in relational databases where data replication is added to one or more tables to avoid costly joins
  - Avoid edge traversals to improve aggregation and 1-hop neighbor lookup in graph queries

Drug (name STRING, brand STRING,
Indication.desc LIST),
Indication (desc STRING),
(Drug)-[treat]->(Indication)



Schema optimization algorithms

- Without space constraints
  - Iteratively apply the proposed relationship rules in order and generate an optimal property graph schema (harness all possible optimization opportunities)
  - **Theorem** – applying the rules in any order results in the same property graph schema [proof in the paper]
- With space constraints
  - Concept-centric algorithm
  - Relation-centric algorithm

Concept-centric (*CC*) schema optimization algorithm

- Core idea – prioritize relationships of key concepts in an ontology
  - Key concepts – similar to PageRank, rank concepts based on ontology structural information
    - / Inheritance and union
    - / Concept out-degree
  - Leverage additional information
    - / Access frequency
    - / Data characteristics

$$Score(c_i) = \frac{\overbrace{c_i.pr}^{\text{OntologyPR score}} \cdot \overbrace{AF(c_i)}^{\text{access frequency}}}{\underbrace{Size(c_i)}_{\text{space consumption}}}$$

Relation-centric (*RC*) schema optimization algorithm

- *CC* algorithm limited to each concept locally (not global optimal)
- Core idea
  - Reduce the relationship selection problem to 0/1 Knapsack problem
    - / Leverage the fully polynomial time approximation scheme (FPTAS) to produce a global optimal solution
  - Prioritize relationships based on a cost-benefit model
    - / Union relationship – $Benefit(r) = AF(c_i \xrightarrow{r} c_j) \mid Cost(r) = \sum_{r' \in (c_i.R_i \setminus R_{union})} |r'|$
    - / Inheritance relationship – $Benefit(r) = AF(c_i \xrightarrow{r} c_j.P_j) \cdot JS(c_i, c_j) \mid$

$$Cost(r) = \begin{cases} \sum_{p \in c_j.P_j} |c_j| \cdot p.type + \sum_{r \in (c_j.R_j \setminus R_{inheritance})} |r|, if\ \theta_1 < JS(c_i, c_j) \\ \sum_{p \in c_i.P_i} |c_i| \cdot p.type + \sum_{r \in (c_i.R_i \setminus R_{inheritance})} |r|, if\ JS(c_i, c_j) < \theta_2 \end{cases}$$

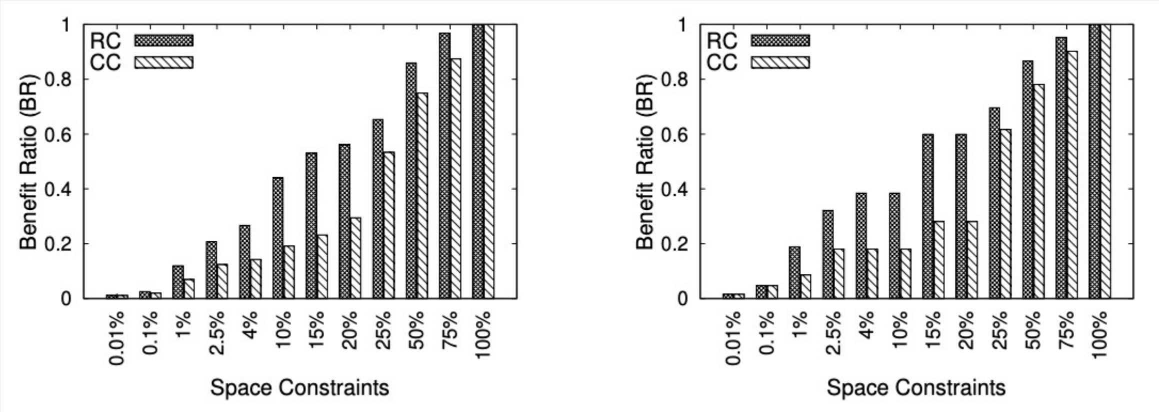
    - / One-to-many and many-to-many relationships – $Benefit(r) = AF(c_i \xrightarrow{r} c_j.P) \mid Cost(r) = |r| \cdot p.type$

Experimental setup

- Two real-world datasets
  - Medical data (`MED`) – 12 GB, 43 concepts, 78 properties, and 58 relationships (11 inheritance, 5 one-to-one, 30 one-to-many, and 12 many-to-many relationships)
  - Financial data (`FIN`) – 53 GB, 90 concepts, 96 properties, and 103 relationships (4 union, 69 inheritance, and 30 one-to-many relationships)
- Two workload summaries (Uniform and Zipf)
- Two graph engines (JanusGraph and Neo4j)
- Measures
  - Property graph schema quality
  - Graph query performance

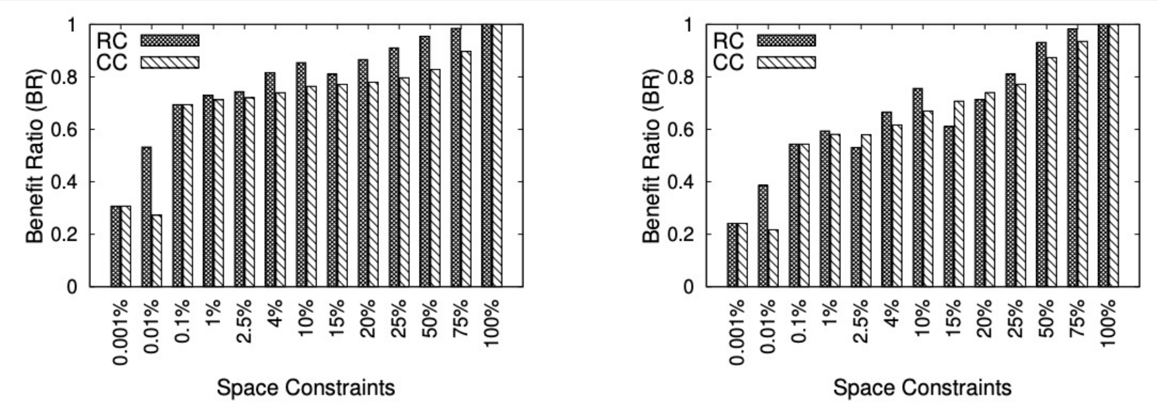
Experimental results – schema quality

- Vary space constraint
  - *RC* consistently outperforms *CC* with both uniform and Zipf workloads
  - Both algorithms effectively utilize the given space constraint



(a) Uniform Workload (b) Zipf Workload

MED

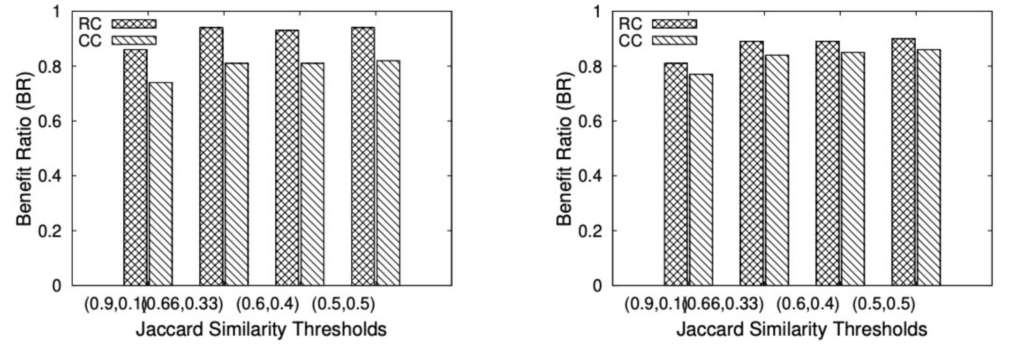


(a) Uniform Workload (b) Zipf Workload

FIN

Experimental results – schema quality

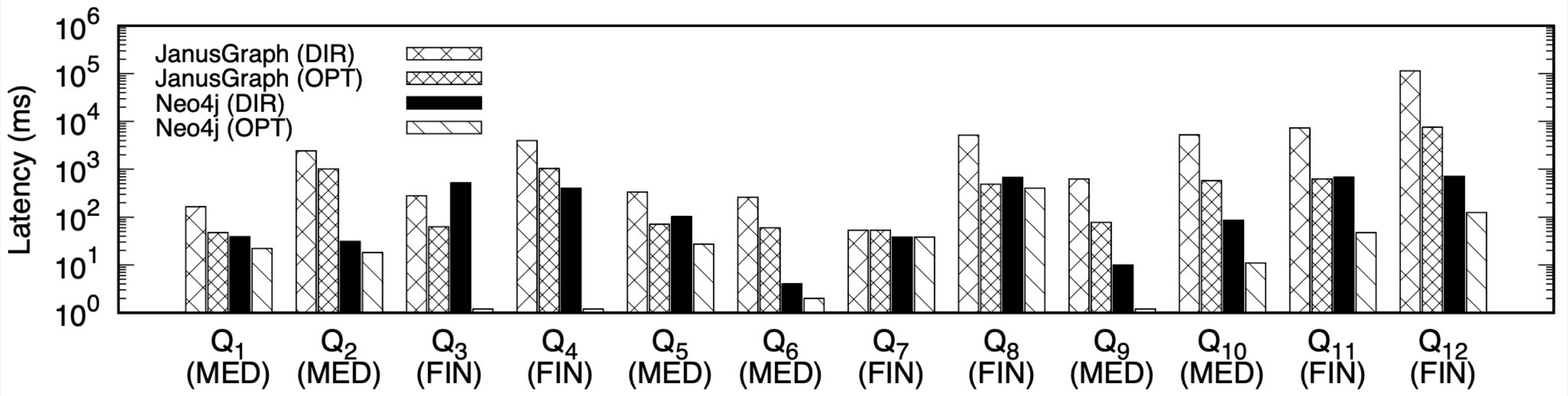
- Vary Jaccard similarity
  - Use FIN as it consists of multiple inheritance relationships
  - Both *CC* and *RC* are robust with different similarity thresholds
  - *RC* outperforms *CC* since it chooses relationships with a global ordering



(a) Uniform Workload (b) Zipf Workload

Experimental results – query performance

Microbenchmark



- Microbenchmark
  - Pattern matching (Q_1-Q_4), property lookup (Q_5-Q_8), aggregation (Q_9-Q_{12})
  - The optimized schema has significant advantages over direct mapping schema for all queries
- Graph query workload
  - The optimized schema offer significant performance boosts to the graph query workloads on both JanusGraph and Neo4j

Conclusions

- Our ontology-driven approach is the first to address the property graph schema optimization
- We propose
  - A set of rules that reduce the edge traversals by exploiting the rich semantic relationships in the ontology, leading to better graph query performance
  - Concept-centric and relation-centric algorithms, utilizing the proposed rules to generate an optimized property graph schema
- Graph queries over the optimized property graphs (`MED` and `FIN`) achieve up to 2 orders of magnitude performance gains compared to the baseline

Thank you!

Questions?