Evaluate knowledge graphs as queryable objects

Introduction

What is a Knowledge Graph (KG)?

A knowledge graph is a specialized form of knowledge base that represents information as a graph structure. It can be defined as a data set that is:

- Structured: Information is organized in a specific data structure
- Normalized: Composed of small units, typically vertices and edges
- Connected: Defined by relationships between objects, which may be direct or distant

Additionally, knowledge graphs are typically:

- Explicit: Created purposefully with intended meaning
- Declarative: Meaningful independently of implementation or algorithms
- Annotated: Enriched with contextual information and metadata
- Non-hierarchical: More complex than simple tree structures
- Large-scale: Often containing millions of elements

Examples of well-known knowledge graphs include Wikidata, Google Knowledge Graph, and OpenStreetMap.

Why do we use it and how?

We use knowledge graphs for several reasons:

- To represent complex, interconnected information in a flexible and scalable way
- To enable powerful querying and analysis of relationships between entities
- To support various applications in artificial intelligence, information retrieval, and data integration

Knowledge graphs are used in various ways:

- As a backbone for search engines and question-answering systems
- To power recommendation systems and personalized content delivery
- For data integration across diverse sources in enterprises
- To support reasoning and inference in AI applications

To work with knowledge graphs, we typically use specialized technologies such as:

- Graph databases (e.g., Neo4j) for storage and querying
- Query languages like SPARQL for RDF-based knowledge graphs
- Ontology languages (e.g., OWL) for defining the structure and rules of the graph
- Graph analysis algorithms for deriving insights and patterns

Proposed method

Key qualities that we care about

- Completedness: All entities are represented.
- Nonredundancy/no miss classified items: If a single entity is counted multiple times, if an out-of-category item is included…
- Accuracy: Properties and relations of entities are correct

How to check these qualities

We can check every (entity, relation, entity) triple with fact (external source or a verification system built on the same text source, like a RAG-chatbot) to ensure accuracy of the KG, but it is not intuitive.
For the particular case of analysing an epic account like *Heike Monogatari*, we can check auxiliary artifacts generated from the KG to assess its quality, namely:

- Family trees for each house and organigrams for each non-secular organization (like a certain temple) to verify genealogical and organizational relationships between characters
- A timeline of important events to validate chronology of occurrences

Experiment

Text source: *Heike Monogatari* (English version)
Generation of KG: LlamaIndex + Neo4j parsing text, extracting and storing knowledge graph
Generation of auxiliary artifact: Query with Cypher…
Evaluation and correction:

(Property Graph, a specific type of KG)

A property graph G is defined as a tuple:G = (N, E, λ, ρ, σ)Where:

- N is a finite set of nodes (vertices)
- E is a finite set of edges (relationships), where each $e \in E$ is an ordered pair (u, v) with $u, v \in N$
- λ is a labeling function that assigns a set of labels to nodes and edges:
  $\lambda : (N \cup E) \to P(L)$, where L is a set of labels and P(L) is the power set of L
- ρ is a property function that assigns a set of key-value pairs to nodes and edges:
  $\rho : (N \cup E) \to P(K \times V)$, where K is a set of keys and V is a set of values
- σ is a type function that assigns a type to each edge:
  $\sigma : E \to T$, where T is a set of relationship types

Key characteristics:

1. It is a directed graph, as edges have a source and target node.

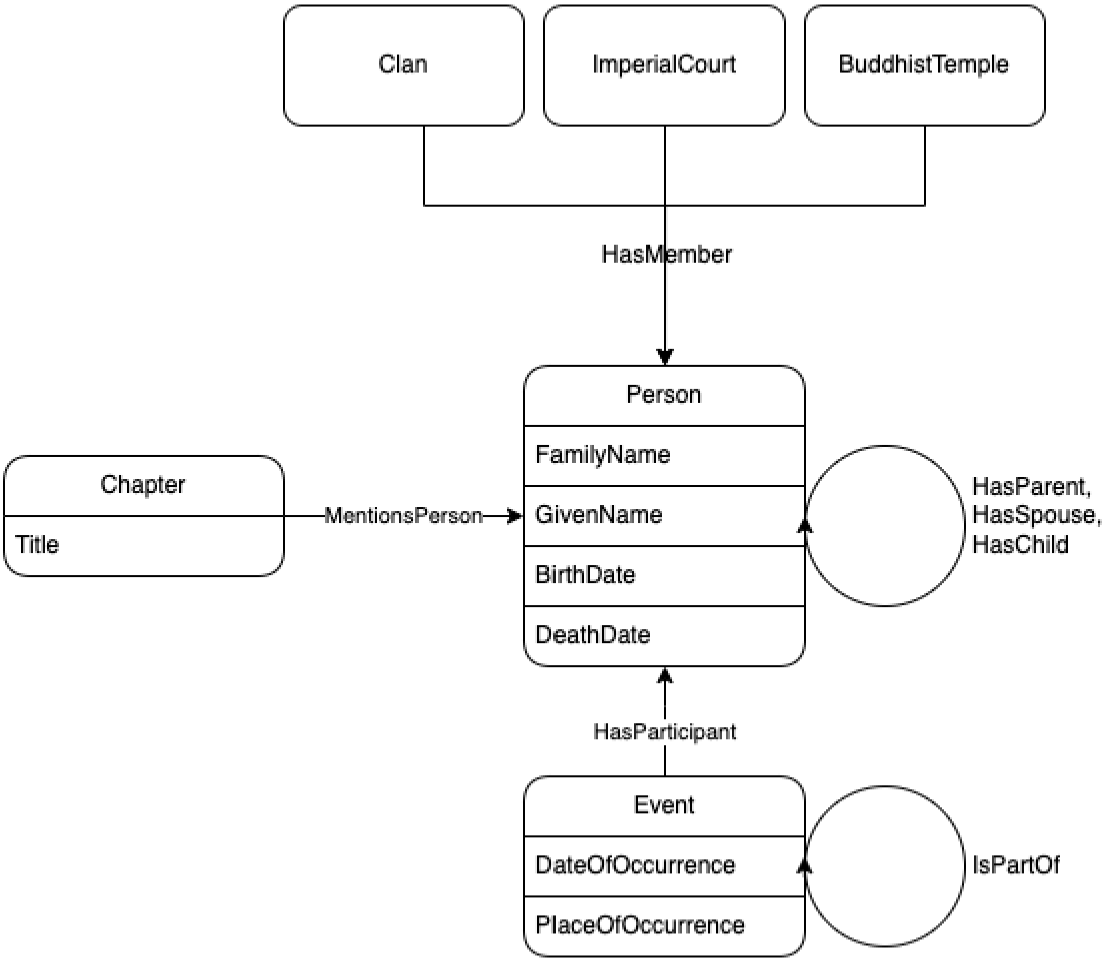
2. It allows multiple edges between the same pair of nodes (multigraph).
3. It permits self-loops (edges from a node to itself).
4. Both nodes and edges can have labels.
5. Both nodes and edges can have properties (key-value pairs).
6. Edges have types.

KG Schema design

We can extract a KG without inital ontology/schema and let the LLM on the fly but It may produce a larger number of diverse relationships but might lack consistency in entity and relation naming[Comparing LLM Path Extractors for Knowledge Graph Construction — Llamaindex]. In order to avoid this, we will provide LLM with with some initial guidance from a schema.
Based on the nature of our source text, a historical account (in medieval Japan) and our expecated functionalities (be able to generate a family tree and a timeline of events) from the KG out of it, we define the KG schema majorly in these categories of pivot nodes:

- Clan (e.g., Taira, Minamoto)
  - hasMember (linking to Person)
- BuddhistTemple
  - hasMember
- ImperialCourt
  - hasMember
- Person (with reference to Family History Knowledge Base and Friend of a Friend ontologies)
  - properties
    - familyName
    - givenName
    - birthDate
    - deathDate
  - relations
    - hasParent
    - ~~hasSibling~~
    - hasSpouse
    - hasChild
- Event (with reference to Event ontology and CIDOC Conceptual Reference Model (for cultural heritage))
  - properties
    - date of occurence
    - place of occurence
    - ~~outcome~~
  - relations
    - hasParticipant (linking to Person)
    - ~~hasOutcome~~

        - isPartOf (for sub-events)
- Chapter (with reference to FaBiO (FRBR-aligned Bibliographic Ontology))
    - properties
        - title
    - relations
        - mentionsPerson (linking text to Person)
        - describesEvent (linking text to Event)



Ontologies are formal, exaustive descriptions of concepts and relationships of which only a partial components are desirable for our application. We try to maintain a simple and clear structure to avoid redundancy such as: “hasGrandParent” can be infered with “hasParent”; [Person]--participates-->[Event] and [Event]--hasParticipant-->[Person] convey the same information.

References

- Source code of LlamaIndex KG extractor with schema validation <https://github.com/run-llama/llama_index/blob/main/llama-index-core/llama_index/core/indices/property_graph/transformations/schema_llm.py>