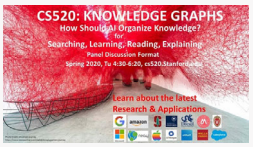

CS520

Knowledge Graphs

What should AI Know ?

8. How to Evolve a Knowledge Graph?

1. Introduction

Once a knowledge graph has been built, it will need to evolve in response to changes in the real-world, and changes in the business requirements. Such changes can be either at the schema level or at the level of individual facts in the knowledge graph. Updating a knowledge graph at the level of individual facts is usually much easier than updating the schema. This is because some changes to schema can have far reaching consequences in the data that is already stored, and sometimes, even in the software that makes assumptions about the schema.

Approaches to handle the evolution of a knowledge graph must address both social and technical challenges. Social challenges need to be addressed for, at least, two reasons. First, design changes inherently have an element of subjectivity, and must have a buy in from the users. Second, as the changes can have impact on the work of multiple stakeholders, suitable workflow processes need to be in place for any change to be rolled out across the user community. Unfortunately, there are not many standards and best practices for how best to handle such social processes.

The technical problems involved in evolving a knowledge graph have been fundamental to database and knowledge base management, and have been researched under the topics of schema evolution, view maintenance, and truth maintenance. Each of these techniques is meant for a different category of updates, and is backed by significant theory that can be adapted for the context of knowledge graphs.

The choice of different techniques for evolving a knowledge graph is also strongly influenced by the business requirements and the cost of making a change. Most large-scale knoweldge graphs have numerous inconsistencies that persist over a period of time, and may be left unaddressed, simply because they do not affect a functionality that is critical for business.

We will begin this chapter by considering several concrete examples of changes that are required in knowledge graph. We will then briefly introduce schema evolution, view maintenance, and truth maintenance. Our goal is not to provide a comprehensive coverage on each of these three topics, but to suggest how these methods are relevant to knowledge graphs, and should be adapted.

2. Examples of Changes to a Knowledge Graph

We will consider examples of the following categories of changes to the knowledge graph: changing world, changing requirements, changing sources, changes affecting previous inferences, and changes requiring redesign. We recognize that there may be more than one way to classify the examples considered here, but we hope that this categorization introduces some structure to the series of examples considered below.

Consider the Amazon product knowledge graph and how it must respond to the changes in the real world. This knowledge graph is constructed by combining information provided by multiple vendors. The product quantities need to be kept up to date with their sales, and as new shipments arrive. Suppliers are constantly providing new products, and some of them may require introducing new properties in the knowledge graph. Some products or product lines may be dropped, and may no longer be necessary.

As an example of changing requirements, consider the concept of *Artist* in Google knowledge graph. The original knowledge graph assumed that an *Artist* can only be a *Person*. Significant checks had been put in the code to ensure that the incoming data satisfied this constraint. But, over a period of time, they started to see data that contained a *Vocaloid* as an artist. As a *Vocaloid* is not a *Person*, and there were a large number of users who cared about listing it as an *Artist*, this assumption needed to be changed.

As an example of the changing sources, consider the problem of creating a knowledge graph for music albums. Many times, the complete information about an album is not available from a single source, and it must be created by combining data from multiple sources. Frequently, new sources need to be added to get a complete picture of an album, and these sources, themselves, change over a period of time. We need to keep the knowledge graph in sync with these changing sources.

As an example of changes that impact inference, consider the constraint that a movie theater shows only movies. A knowledge graph may infer that any program that is being shown in a movie theater is a movie. But, more recently, sporting events, and sometimes, operas are shown in a movie theater. Based on the constraint that a movie theater shows only movies, the inference algorithm will incorrectly conclude that a sporting event or an opera is also a movie. Fixing this requires updating the relevant inference rules that draw these conclusions.

As an example of a change requiring redesign, consider the situation where a knowledge graph initially represents the *CEO* of a company as a relation that has a *Person* as a value. The designers may choose to redesign this representation so that the value of the *CEO* relation is another object with properties such as the time period for which that person held that position.

3. Schema Evolution Techniques

Schema evolution for a relational database is referred to as database reogranization, and addresses problems that arise when adding/removing a column. Schema evolution for a knowledge graph is more complex, and has operations such as adding/removing a class from the class hierarchy, adding/removing a superclass to an existing class, adding/removing type of an individual, adding/removing a relation or a property from a class. The additional complexity arises because the class hierarchy needs to satisfy some constraints, and some of the information inherits and propagates through the knowledge graph. We will next discuss some of these intricacies in greater detail.

When we remove or rename a property, this change must be propagated throughout the knowledge graph. It is typical to generate a summary of all affected places where such an update will impact the knowledge graph.

When we add a new class to the class hierarchy without specifying its immediate superclass, we can assume it to be the class of a system-defined root class. This assumption is based on the constraint that we will not like to have orphan classes in the class hierarchy.

When we delete a class from the class hierarchy, or remove the superclass of a class, we need to make decisions about what to do about its subclasses, and its instances. If its subclasses have another superclass, this does not pose a problem. But, if the class being deleted is the only superclass of a class, we need to either delete all the subclasses, and their instances or we need to assign a new superclass. To assign a new superclass to class A whose only superclass B is being deleted, the new superclass could be one or all immediate superclasses of B. Furthermore, if there was any property associated with the class that is being deleted, we need to either delete those properties from all the classes and instances where it was inherited, or ensure that the property is associated with another class that will remain in the knowledge graph after the deletion of the class for which this property was originally defined.

Some updates to a class hierarchy can create subtle situations. Consider a situation in which A is a superclass of B, and B is a superclass of C. Suppose we assert A to be a direct superclass of C. As

this relationship can also be inferred through transitivity, should such an update even be allowed? Many knowledge graph systems will reject such an update to the class hierarchy.

When adding a new superclass relationship to a class hierarchy, we need to make sure that we maintain the acylicity in the class hierarchy. If adding a new class will create cycles, support should be provided to detect and address the source of the problem.

When we change the constraint on a relation, we can run into at least two different situations. If we are relaxing the constraint, then it does not affect the existing data. But if we make the constraint on a relation tighter, it may invalidate some existing data, and suitable repair actions will be required.

4. View Maintenance Techniques

Views are a mechanism in relational database management systems to name a query so that it can be executed simply by referencing the name. The query is defined with respect to a set of one or more tables, referred to as the *base* tables. If we choose to store the results of the query corresponding to a view, it is referred to as a *materialized* view. If a view is materialized, and there is a change to data in the base tables, the view must be updated. Even though the simplest approach is to recompute the view from scratch, several efficient algorithms, known as *incremental* view maintenance algorithms are available that do not compute the view from scratch.

In the context of knowledge graphs, the use of view computation and maintenance techniques is currently not very common. There are situations where data in the knowledge graph is processed, and the results are stored. For such situations, view definition and maintenance techniques are directly applicable. Leveraging view update methods for such situations is open for future work.

5. Truth Maintenance Techniques

Truth maintenance techniques were originally developed in the context of rule-based systems to keep track of the derived conclusions which can be updated in response to any changes in data or rules. A popular implementation of a truth maintenance system is known as a justification system. In a justification-based system, every time a new piece of information is derived, the system records a justification for the derivation. The justification often includes other facts, and rules that were used during the derivation. At a future time, whenever there is a change in a fact or a rule, one can examine various derivations where that fact or rule appears, and update those derivations by taking the new information into account.

In the context of knowledge graphs, the inferences are usually derived by the application code. In the current state of knowledge graph practice, the derived inferences are not explicitly tracked. As the modern graph engines mature, we anticipate there to be need for explicitly tracking the source of an inference to enable efficient update mechanisms.

6. Summary

Knowledge graphs are created in response to specific business needs, and have a life cycle. Many knowledge graphs persist over a long period of times, and must evolve in response to the changes in the real-world, and the changes in the business requirements. The evolution of a knowledge graph needs to be sensitive to its usage by the user community, needs to follow well-defined engineering processes, and can benefit by using design principles and algorithms. The design principles and algorithms can leverage from the past work on schema evolution, view maintenance, and truth maintenance system which will be increasingly important in the future generation of knowledge graph system.

Exercises

Exercise 8.1. Assess the qualitative ease (ie, easy, moderately involved, highly involved) of

making the following changes in a knowledge graph.

(*a*) Incorporating release of a new iPhone into Amazon product graph

(*b*) Incorporating the effect of Brexit

(*c*) Launch of a new vendor for distributing face masks

(*d*) Repurposing a hotel as a hospital for COVID patients

(*e*) Changing Wikipedia knowledge graph to model corporate mergers and acqusitions

Exercise 8.2. For each of the following change, which of the knowledge graph change management technique would be most directly applicable? (Recall that the change management techniques are: schema evolution, view maintenance and truth maintenance).

(*a*) Eliminating a product category

(*b*) Wiki Data integrates data from different museums, city governments, and bibliographic databases

(*c*) Evolving Microsoft academic publications knowledge graph

(*d*) Updates in the schema mapping rules

(*e*) Splitting a category into two different categories