Graph Summarization for Entity Relatedness Visualization

Yukai Miao
The University of New South Wales
Australia
yukai.miao@unsw.edu.au

Jianbin Qin*
The University of New South Wales
Australia
jqin@cse.unsw.edu.au

Wei Wang
The University of New South Wales
Australia
weiw@cse.unsw.edu.au

ABSTRACT
In modern search engines, Knowledge Graphs have become a key component for knowledge discovery. When a user searches for an entity, the existing systems usually provide a list of related entities, but they do not necessarily give explanations of how they are related. However, with the help of knowledge graphs, we can generate relatedness graphs between any pair of existing entities. Existing methods of this problem are either graph-based or list-based, but they all have some limitations when dealing with large complex relatedness graphs of two related entity. In this work, we investigate how to summarize the relatedness graphs and how to use the summarized graphs to assist the users to retrieve target information. We also implemented our approach in an online query system and performed experiments and evaluations on it. The results show that our method produces much better result than previous work.

CCS CONCEPTS
• Information systems → Information systems applications; Information retrieval;

KEYWORDS
Graph Summarization; Graph Visualization; Knowledge Graph

1 INTRODUCTION
Nowadays, search engine companies like Google and Yahoo! usually provide a knowledge card about the queried "things" besides the traditional list of "blue links." For example, Figure 1 shows part of a knowledge card provided by Google when we search for "Einstein". As we can see, some recommended people have text labels to indicate their relationships to Einstein, but the others do not have any descriptions about why they are related. In particular, "Isaac Newton" is the top-ranked entity, but we know that he is not very "close" to Albert Einstein, and their relationships are indeed hard to explain by a single word.

However, with the help of some public knowledge graphs, e.g., DBPedia, we can find some paths that connect them. Then, we use the combination of these paths to represent their relationship and visualize them to retrieve deep information.

Currently, there are two ways to do this. They are the graph-based approach and the list-based approach. In the graph-based approach [3, 6], all the relevant information extracted from the knowledge base are represented as a single graph. This approach gives users the overview of the relationship, but when the graph goes bigger, it may be too complex for humans to navigate through it and get some findings. To solve this complexity problem, they need some filtering methods to reduce the graph size. On the other hand, the list-based approach [1, 2] generates a ranked list of path patterns or subgraph patterns from the knowledge graph. This approach directly shows the important information to the users but it breaks the overall structure and the users cannot navigate through the edges easily. Besides, a top-K list will eliminate the tail information.

To overcome the limits of the previous approaches, we propose a novel method to tackle the relatedness visualization problem. We adopt the general framework of graph-based approach but enhance the expressiveness of the graphs via a summarization method. Firstly, given a pair of query entities, we use the existing methods to extract the relatedness information from the knowledge base. Then, we preprocess it with a simple heuristic to do a decent graph reduction that only removes redundant information. Finally, we use a classical model, Bisimulation, to summarize (or simplify) the graph into a more concise form. For the summarization, we allow the users to adjust the parameters online and generate the resulting graph instantly. In other words, our summarization method takes a pair of query entities, their preprocessed relatedness graph and the users’ configurations as the input, and computes a summarized graph as the output. The advantages of this approach are

• Avoid the redundant information propagated from the intermediate entities.
• Keep all the non-redundant information intact.
• Keep the high-level structures and hide the low-level details as user specified.
• Visualize the summarized graph to support easy navigation.

In this work, our goal is to use summarized graphs to visualize relatedness graphs effectively and efficiently. The main contributions of this work are

• This is the first work to apply a classic model, Bisimulation, to summarize (or simplify) the relatedness graphs for visualization.
• We design a graph summarization approach to help users investigate a complex relatedness graph.

2 RELATED WORK
We firstly introduce two major categories of approaches for this problem, and then give some background about Bisimulation.
Within a relatedness explanation, an edge is necessary if and only if it belongs to a simple path\(^1\) between the source entity \(v_s\) and the target entity \(v_t\). If a relatedness explanation contains only necessary edges, then it becomes a necessary relatedness explanation.

For example, in Figure 2, there is a relatedness explanation of query \((S, T)\), in which all edges except \(r\) are necessary edges. So, if we remove \(r\) and \(C\) (because \(C\) is disconnected from the explanation after \(r\) is removed), then the remaining subgraph becomes a necessary relatedness explanation.

---

\(^{1}\)The edges can be directed or undirected, but we may treat each undirected edge as two directed edges in opposite directions, then the whole graph becomes a directed graph.

---

3 PRELIMINARY

3.1 Knowledge Graph

A Knowledge Graph (or KG) is a graph that consists of entities (e.g., persons, locations, and organizations) as nodes and relations between pairs of entities (e.g., "spouse", "bornIn", and "memberOf") as edges\(^2\). A knowledge graph can be formally defined as below.

**Definition 3.1 (Knowledge Graph).** Knowledge Graph \(G\) is a triple \(G := (V, E, \lambda)\) where \(V\) is the set of nodes, \(E\) is the set of edges, and \(\lambda := E \rightarrow L\) is the edge labeling function that gives each edge a label \(r \in R\).

Usually, given an edge \(s \xrightarrow{r} o\), we call \(s\), \(r\), and \(o\) as "subject", "relation type" and "object", respectively.

3.2 Relatedness Explanation

**Definition 3.2 (Relatedness Explanation).** Given a knowledge graph \(G\) and a pair of query entities \((v_s, v_t)\), the Relatedness Explanation is a triple \((v_s, v_t, G_{r})\), where \(G_{r}\) is a subgraph of \(G\).

---

\(^{2}\)Because the direction of each edge only represents the semantic meaning of that relation, regardless of which direction it is, the two nodes connected by this edge are related. So we do not care about directions when searching for paths.

\(^{3}\)Informally, a refinement of a partition \(P\) is a further partition of \(P\), where some blocks of \(P\) split into smaller blocks. If \(Q\) is a refinement of \(P\), then \(P\) is coarser than \(Q\). Due to the page limit, please find some related materials for the more details.

---

Figure 1: Related Entities suggested by Google when searching for "Albert Einstein"

**Graph-based:** RelFinder [3] is one of the early work on relatedness extraction and visualization. It is a graph-based method to visualize the relatedness graph. To handle large graph, RelFinder requires fine-tuned filtering predicates. RECAP [6] is one of the state-of-art relatedness extraction and visualization system. It uses statistical path ranking methods to generate top related paths. Its final relatedness graph is the combination of the top paths.

**List-based:** REX [2] is one of the early work on list-based relatedness extraction. It enumerates graph patterns with graph-level or path-level enumeration approaches and then directly performs ranking on these graph patterns. Exprax [1] is another list-based system for relatedness queries. It utilizes the type hierarchy of nodes and edges to generate different levels of patterns and transforms the top-k ranking problem into an optimization problem.

**Bisimulation:** In terms of Bisimulation, [3] gives an \(O(m \log(n))\) time complexity algorithm to compute the coarsest stable partition in a graph with \(n\) nodes and \(m\) edges. We adapt this algorithm for our graph simplification task. Besides, [4] shows us the application of bisimulation for building indices and querying efficiently in XML.
4 PREPROCESSING

The initial step of the relatedness explanation is to extract a subgraph that connects two query entities. We borrow the previous methods that transform a relatedness query into a path enumeration problem. It searches for all the simple paths, with a maximal length restriction, that start from one of the query entities and terminate at the other one.

Among the paths we enumerated, some paths can be redundant and we want to remove them. Given a set of candidate paths $H$, a path $h \in H$ is redundant if it contains a sub-path $s$ such that if we cut out $s$ from $h$ and connect the remaining two parts, the resulting path $h'$ is also a candidate in $H$. For example, given a pair of query entities $(v_i, v_j)$, we get two simple paths, $h_1$ and $h_2$ as below.

$$h_1 = v_k - \cdots - v_i - v_{i+1} - \cdots - v_l, \quad h_2 = v_k - \cdots - v_i - u_1 - \cdots - u_y - v_{i+1} - \cdots - v_l$$

The only difference between $h_1$ and $h_2$ is the middle part. Since $v_i$ and $v_{i+1}$ are directly connected in $h_1$, we find that sub-path $z = u_1 - \cdots - u_y$ is redundant as it also connects $v_i$ and $v_{i+1}$. So, we remove path $h_2$ to keep the graph concise.

In the final step, the paths are merged into a relatedness graph, and we will use it as the initial graph of our summarization method.

5 GRAPH SUMMARIZATION

After we extract and preprocess a relatedness graph, we need a method to summarize it and then visualize it. Our general idea is to use the maximal bisimulation to partition the given graph and generate a summarized graph using the partitioned blocks. The maximal bisimulation problem is well-defined and it has a fixed result given an input graph and its initial partition. However, the users may have different focuses on the graph in different scenarios, so we need to take the user-defined configurations as the parameters in the computation. In the following parts, we will show how to adapt the bisimulation to our problem and how to support two kinds of user-defined predicates, i.e. entity type predicates and relation type predicates, to generate a properly summarized graph.5

Adapt Bisimulation to Relatedness Graphs. The original definition of bisimulation only deals with non-labeled graphs, but a relatedness graph is a labeled graph, where the labels are relation types. To take these relation types into account, we extend the bisimulation definition below.

If $x \xrightarrow{p} x'$, then there is some $y \xrightarrow{q} y'$ such that $x' \xrightarrow{R} y'$. \hspace{1cm} (1)

If $y \xrightarrow{q} y'$, then there is some $x \xrightarrow{p} x'$ such that $x' \xrightarrow{R} y'$.

Support Entity Type Predicate. In a specific task, for example, we may focus on only "persons" and "organizations" and we do not care about the other types of entities. Thus, we expect that these types of entities are distinguished from other types of entities in the summarized graph. To support summarization with such requirement, we have to change the initial partition $P$. We first start by divide the initial partition into three sets, i.e. $B_{\text{person}}, B_{\text{org}}$ and $B_{\text{others}}$. Then, we use $P = \{\{v_j\}, \{v_i\}, B_{\text{person}}, B_{\text{org}}, B_{\text{others}}\}$ as the initial partition of the bisimulation. According to the definition of the maximal bisimulation problem, given this initial partition $P$, the final partition must keep the selected types of entities, i.e. "persons" and "organizations", separated from all the other types of entities.

Support Relation Type Predicates. Similarly, we may be interested in only some of the relation types for a particular task, so we want to respect the definition of bisimulation only on these edges. To support this, we create an "activated edge set" $R_a$ and use it as one of the parameters of bisimulation. For example, if the user selects relation type BirthDate as the predicate. All edges with label BirthDate will be added into $R_a$. Then the modified bisimulation algorithms based on Formula (1) will compute the maximal bisimulation considering only edges in $R_a$. Based on the definition of bisimulation, any pairs of nodes which are not bisimilar in terms of edges in $R_a$ must be partitioned into the different partitions.6

Example. We show an example of how the initial graph is simplified in the figure 3. In this example, the two query entities are "Frank Herbert" and "Brian Herbert". Figure 3(a) shows an initial relatedness graph generated from the methods in Section 4. Figure 3(b) shows the summarized graph after we apply maximal bisimulation without defining any semantic predicates on both entity type and relation type. However, this graph is too concise to extract any target information. In figure 3(c), we show a meaningful summarization graph with entity type predicate "writer" and relation type predicate "author". We can use this graph to retrieve some information such as "Which book was co-authored by them?".

6 EVALUATION

6.1 Evaluation Setup

To evaluate our proposed approach, we implemented a system called REVS. We conducted a user-centered evaluation to compare our system with other similar systems for relatedness extraction and visualization. We used RelFinder [3] as a representative of graph-based approach and Explass [1] as a representative of list-based approach. For the evaluation, we created 20 questions involving 10 pairs of entities and invited 15 persons to use these systems to find the answers. These questions are based on 2 one-hop relations, 12 two-hops relations and 6 three-hops relations.7 We collect the answers, time to complete each answer and users’ rating to each system for each question.

Before the evaluation starts, we did a small experiment on the effect of summarization by ourselves. For each question, we tried to find the best configuration that produces the simplest graph and also exposes the answer8, and we recorded the amount of nodes $N$ and predicates $E$ of the graph in each phase.

6.2 Evaluation Results and Analysis

Effect of Summarization. The amount of nodes and edges in each phase for some questions are shown in Table 1. In this table, each

---

5In most cases, the maximal path length is set to be 3.

6A big difference between our method and other existing methods is that we use types to affect the partition rather than filtering out some edges and nodes.

7The source code and experiment resources are available at https://github.com/DBWangGroupUNSW/revs.

8Typically, we just enable only the predicates and entity types that exist in the questions.

---
As we can see, the preprocessing does reduce the initial graph complexity, but its performance depends on the internal structures of the original graphs. For example, Q1 gets the most benefit from the preprocessing step because many sub-paths that connect some pairs of “popular” neighboring entities are redundant and their direct relations have already provided enough information. However, Q9 gets almost no help from it because nearly all the internal relations do not contain such redundancy. Nevertheless, as shown in the last two columns, after the bisimulation is applied, the resulting graph are heavily reduced. For all the 20 questions, no more than 10 nodes are required to show the answers.

User Study: The results of the evaluation are shown in Table 2. Firstly, we have the correctness rates $C$ of each system for different groups of questions. Secondly, we compute the scores $S = \frac{\log(r + 1)}{t}$ for each system, where $r$ is users’ rating and $t$ is the time cost, to measure how well they help the users find the answers. If a participant gives a wrong answer to a question, we treat this as a 0-rating case.

In general, REVS performs best in all the question groups. Both REVS and Explass beat RelFinder because they provide summarized information. Expllass looks bad at handling 1-hop relations because it always put this kind of relations into the “other paths” list. For multi-hop relations, REVS has better performance than Explass because REVS provides a global view of all the relatedness information and it gives users a direct impression of where the answer could be.

7 LIMITATION & FUTURE WORK

In this work, we apply Bisimulation to summarize the relatedness explanations and visualize them to help users retrieve the target information. Due to the lack of reliable benchmark baselines, we designed our own user-centered analysis. We will do further study on the effectiveness of this method and get more insights about its strengths and weaknesses.

REFERENCES


