Entity-driven Type Hierarchy Construction for Freebase

Jyun-Yu Jiang†[*], Chin-Yew Lin‡ and Pu-Jen Cheng†
†Department of Computer Science and Information Engineering, National Taiwan University
‡Microsoft Research Asia
jyunyu.jiang@gmail.com, cyl@microsoft.com, pjcheng@csie.ntu.edu.tw

ABSTRACT

The hierarchical structure of a knowledge base system can lead to various valuable applications; however, many knowledge base systems do not have such property. In this paper, we propose an entity-driven approach to automatically construct the hierarchical structure of entities for knowledge base systems. By deriving type dependencies from entity information, the initial graph of types will be constructed, and then modified to become a hierarchical structure by several graph algorithms. Experimental results show the effectiveness of our method in terms of constructing reasonable type hierarchy for knowledge base systems.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

Keywords

Knowledge Base; Entity Type; Type Hierarchy.

1. INTRODUCTION

Modern knowledge base systems record abundant information about myriads of entities. Most of them exploit entity types to categorize entities. For example, each entity in Freebase [2] has several types. However, entity types of many knowledge base systems are not hierarchical, such as Freebase [2][1] and DBpedia [1], so dependencies among different entity types are also unable to be interpreted. In other words, the capability of entity types can be improved with the hierarchical information. For instance, `film.actor` and `people.person` are two types in Freebase. If the dependency between two types is provided, we can infer that an entity with the type `film.actor` must belong to the type `people.person` because `people.person` is an ancestor of `film.actor` in the type hierarchy. Therefore, an approach to automatically construct type hierarchy is needed for improving knowledge base systems.

In this paper, given entities with their types in a knowledge base system, we aim to construct the hierarchical structure for such entity types. For example, given entities with types `people.person`, `sports.athlete` and `tennis.player`, we would like to derive the hierarchical structure as follows:

`people.person` ← `sports.athlete` ← `tennis.player`,

where the type `people.person` is the root of the hierarchy for human entities. One possible solution is to find identical entities in other hierarchical knowledge base systems, and then align the types [3,4]. However, there are some critical problems. First, some entities and types might not exist in other knowledge base systems, so they cannot be aligned and transferring information. Second, the alignment of entities is not actually equivalent to the mapping of entity types, so the hierarchy of the other system cannot be directly transferred. Hence, in this work, our goal is to construct the hierarchical structure of entities types with only the entities in the original knowledge base system.

To solve the problem, we propose a data-driven approach to construct the type hierarchy for knowledge base systems without hierarchical structure, such as Freebase [2]. The principal idea of our approach is that entities of a type t_1 may contain the other general type t_2, which should be the ancestor of t_1. Exploiting entities as hints, we mine potential dependencies among types. After deriving potential dependencies, several graph algorithms are applied to adjust dependencies to become hierarchical. Finally, experimental results show that our approach can well build satisfactory type hierarchies for the Freebase knowledge base system.

Algorithm 1 Initial Graph Construction

```
Input: Given entity set: $E$; type set: $T$; threshold of dependency confidence: $\theta$ ($0 \le \theta \le 1$).
Output: Initial graph: $G_\theta = (V, A_\theta)$, where $V = T$ and $A_\theta$ is the set of edges (i.e., dependencies among types).
1: $V \leftarrow T, A_\theta \leftarrow \emptyset$
2: for $t_1 \in T$ do
3:     for $t_2 \in T - \{t_1\}$ do
4:         if $|\mathbb{E}(t_1) \cap \mathbb{E}(t_2)| \,/\, |\mathbb{E}(t_1)| \ge \theta$ then
5:             $A_\theta \leftarrow A_\theta \cup \{(t_1 \to t_2)\}$
6:         end if
7:     end for
8: end for
9: return $G_\theta = (V, A_\theta)$
```

2. PROPOSED METHOD

Given the type set T and the entity set E, we aim to automatically construct a hierarchical structure G for all types $t \in T$. Denote $\mathbb{E}(t) \subseteq E$ as the set of entities with the type t. If most of entities in $\mathbb{E}(t_1)$ belong to the type t_2, the type t_1 may be dependent to t_2. Based on this observation, we can derive several potential dependencies among types and construct the hierarchy. Therefore, we propose an entity-driven approach consisting of four stages: (1) initial graph construction, (2) equivalent type contraction, (3) back-edge elimination and (4) vertex separation with cross-edges.

(1) Initial Graph Construction. We first derive the potential dependencies among entity types from entity information as shown in Algorithm 1. By verifying the proportion of shared entities for each type pair, the confidence of dependencies can be identified. Here θ ($0 \le \theta \le 1$) is a threshold to sieve out confident enough dependencies, and G_θ is a directed graph. Only the dependencies to types with more than θ of shared entities will be kept in the graph as edges. However, such initial graph may not be a hierarchical structure. The relationship among entity types will be too complicated to understand. Hence, the graph requires adjusting and modifying to become hierarchical.

[*]Part of this work was done when the first author was on an internship at Microsoft Research Asia

[1]Although every entity type in Freebase belongs a brief group called the type domain, there is still no hierarchical structure.

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742737.

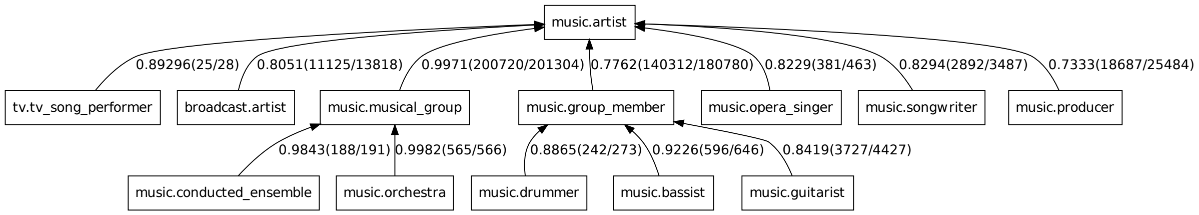

Figure 1: The sub-hierarchy rooted by the type `music.artist` while $\theta_L = 0.7$. Numbers beside edges are the confidence of edges and the detailed proportions of shared entities.

(2) Equivalent Type Contraction. A problem is that the graph may contain cycles, which represent the dependencies propagate back to the types themselves. In the case of edges with high confidence, a cycle may show that types are equivalent because they are likely to be dependent to each other. To put it differently, they can be contracted into a single vertex to avoid the cycle and represent the same idea.

To identify types in cycles, Kosaraju's algorithm [5] is applied to find maximal strongly connected components (SCCs). The graph theory guarantees that vertices must be in a SCC if and only if they are in a cycle. Given a higher threshold θ_H and the corresponding graph G_{θ_H}, we contract vertices in each SCC into a vertex, and then denote the new vertex set as V_{θ_H}. Next, the graph $G_{\theta_H,\theta_L} = (V_{\theta_H}, A_{\theta_L})$ is constructed by the contracted vertex set V_{θ_H} and the edge set A_{θ_L} with a lower confidence threshold θ_L. However, cycles might still exist in A_{θ_L}, so we build G'_{θ_H,θ_L} by iteratively finding the largest cycle, and then removing the edge with the lowest confidence until there is no cycle. In this paper, we set θ_H as 0.9 and θ_L as a tunable parameter. Finally, the graph G'_{θ_H,θ_L} will be a directed acyclic graph.

(3) Back-edge Elimination. While constructing the graph, some edges, especially back-edges, might be redundant for the hierarchical structure. For example, if there are three edges $(t_1 \to t_2)$, $(t_2 \to t_3)$ and $(t_1 \to t_3)$, the back-edge $(t_1 \to t_3)$ is redundant because t_1 can trace its ancestors to t_3 without $(t_1 \to t_3)$. Here we apply the simple depth-first search to find and eliminate all back-edges in the graph.

(4) Vertex Separation with Cross-edges. Since all pairs of types will be verified in Algorithm 1, it is possible that a type is dependent to multiple types. It is reasonable because an entity may act as different characters. For example, a US president is not only a politician but an appointer of the government, so the type `government.us_president` should be dependent to `government.politician` and `people.appointer` at the same time. To solve this problem, for a vertex with k out-links in the graph (i.e., a type dependent to other k types), we separate the vertex and its descendants into k different sub-trees below k corresponding vertices. Hence, in the graph, each vertex will have at most only one parent.

Finally, the graph is a forest consisting many hierarchies about specific topics. We then create a pseudo root r, and let all roots of hierarchies link to r. Therefore, the type hierarchy of the knowledge base system is well constructed.

3. EXPERIMENTS

We utilize the Freebase dataset[2] from December 2014, including more than 90 millions unique entities and 1,950 entity types in more than 80 domains.

Table 1 shows four examples of contracted equivalent entity types. The first three examples are much reasonable.

[2]Freebase Datadumps: https://developers.google.com/freebase/data

Table 1: Examples of equivalent types with $\theta_H = 0.9$.

1. `aviation.airline` $\leftrightarrow$ `aviation.aircraft_owner`
2. `astronomy.discovery` $\leftrightarrow$ `astronomy.star_system_body`
3. `book.scholarly_work` $\leftrightarrow$ `education.dissertation`
4. `music.release` $\leftrightarrow$ `media_common.creative_work`

Table 2: The accuracy and entity coverage of edges in the graphs with different θ_L.

θ_L	0.5	0.6	0.7	0.8	0.9
Accuracy	0.8936	0.9033	0.9124	0.9213	**0.9414**
Coverage	**0.9369**	0.9274	0.9209	0.9167	0.8908

For instance, an airline must be the aircraft owner, and vice versa. The last example is stranger than others because a creative work might belong to not only music but also diversified media. This is caused by the entity bias. Most of creative works in Freebase are in the form of music, so an incorrect edge from `creative_work` to `music.release` exists.

Figure 1 shows an example of the sub-tree while $\theta_L = 0.7$. Entity types are well arranged into a hierarchy by appropriate levels. To further evaluate the quality of the hierarchy construction, we annotated the correctness of each edge in the hierarchies with different θ_L, i.e., the accuracy of edges. Moreover, we evaluate the entity coverage of the built hierarchies. Table 2 shows the accuracy and the coverage of edges in the hierarchy. Generally, the accuracy of every setting is satisfactory. It is obvious that larger θ_L leads to higher accuracy and lower coverage. This is because the confidence of type dependencies increases, but less edges pass the threshold. We then analyze the error in the hierarchy construction. The major mistakes are still about the entity bias. For example, an incorrect edge is (`military.commander` $\to$ `people.deceased_person`) because more than 90% of commanders in Freebase have passed away.

4. CONCLUSION AND FUTURE WORK

In this paper, we focused automatically constructing the type hierarchy for knowledge based systems without hierarchical structure. We propose an entity-driven approach to exploit entity information to derive the potential dependencies and build the type hierarchy. Experimental results shows the effectiveness of our method. The ongoing work is to apply the type hierarchy of Freebase to mine knowledge in other systems, such as web search engines.

5. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08*, pages 1247–1250. ACM, 2008.

[3] E. Demidova, I. Oelze, and W. Nejdl. Aligning freebase with the yago ontology. In *CIKM '13*, pages 579–588. ACM, 2013.

[4] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *WWW '02*, pages 662–673. ACM, 2002.

[5] S. R. Kosaraju and G. Sullivan. Detecting cycles in dynamic graphs in polynomial time. In *STOC '88*, pages 398–406. ACM, 1988.