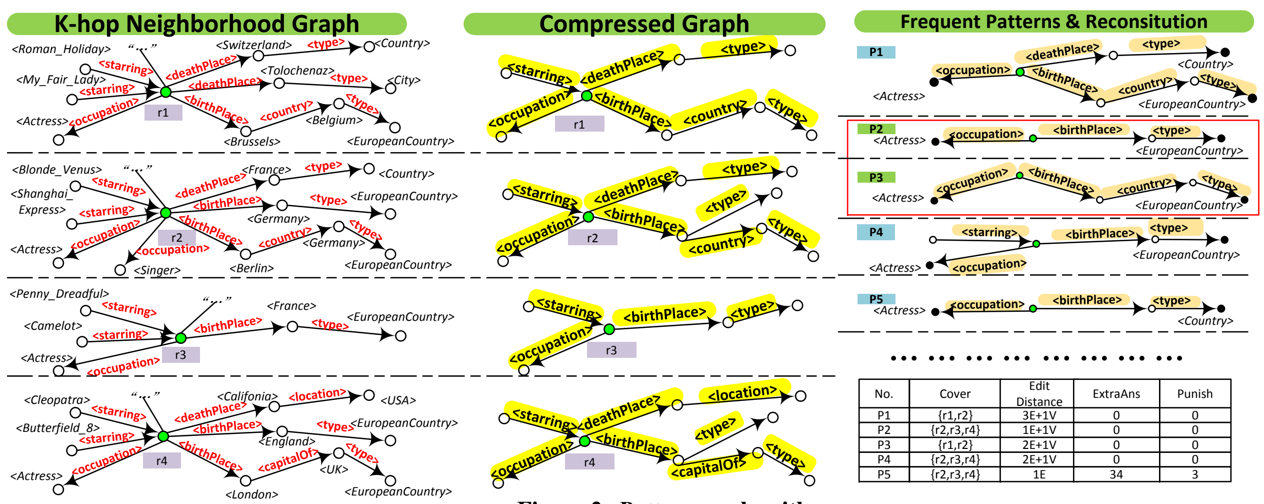

No.	Cover	Edit Distance	ExtraAns	Punish
P1	{r1,r2}	3E+1V	0	0
P2	{r2,r3,r4}	1E+1V	0	0
P3	{r1,r2}	2E+1V	0	0
P4	{r2,r3,r4}	2E+1V	0	0
P5	{r2,r3,r4}	1E	34	3

Figure 2: *Bottom-up* algorithm.

user, excludes the wrong answers $Q^-(D)$ indicated by the user, that is, $\mathcal{R} = Q^+(D) \cup Q^\Delta(D)$, and $Q^-(D) \cap \mathcal{R} = \varnothing$. Our goal is to get a refined query $\mathcal{Q}' = Q'_1 \cup Q'_2 \cup \ldots \cup Q'_s, s \geq 1$, which can cover all the answers in $\mathcal{R}$ and exclude the answers in $Q^-(D)$, also minimize the edit distance between Q and $\mathcal{Q}'$.

The proposed system framework is separated into two parts: *Online Processing* and *Log Dealing*. *Online Processing* stage implements a *Bottom-up* method and a *Candidate-selection* method to ensure the expected modification of this query. And we design a *Rule-Mining* algorithm in *Log Dealing* stage to refine the existing entity-mapping dictionary and relation-paraphrase dictionary, and also build up a sentence-structure dictionary, which will improve existing Q/A systems and subsequent question-answering.

Online Processing During this stage, the new answer set $\mathcal{R}$ after assessments of the user is fed into the *Bottom-up* algorithm. After obtaining candidate graphs, we take them along with Q as input to get the refined SPARQL through the *Candidate-selection* method.

Bottom-up: This part starts from new answer set $\mathcal{R}$ and captures the common feature of accepted answers.

(1) *Get Neighborhood Graphs.* For each answer in $\mathcal{R}$, we find the subgraph induced by its k-hop neighbors.

(2) *Get Compressed Graphs.* We design a method to compress the size of neighborhood graphs. First, we find all simple paths starting from each answer to change the graph into a tree. Then, ignore labels of entities and we can figure out that many paths share same edge label. Insert those paths into a prefix tree, so redundancy vertices and edges are removed. The original neighborhood graphs are compressed obviously while their main structures are still kept.

(3) *Mine frequent patterns.* We devise to extract the common patterns among the Compressed Graphs, so we can deduce the expected query graph. We annotate the nodes on the tree with the label of level, and mine frequent patterns from all these trees using *gSpan* described in [1], altogether denoted by set P.

Figure 2 shows how to apply *Bottom-up* on the example we mentioned in Figure 1, where $Q^+(D)$={r_1,r_2}, $Q^-(D)$={n_1, n_2, n_3}, $Q^\Delta(D)$={r_3,r_4}, and $\mathcal{R}$={r_1,r_2,r_3,r_4}. It displays 3-hop Neighborhood Graphs of answers in $\mathcal{R}$. Take r_2 as an example. Merge paths sharing the same prefix label ⟨*birthPlace*⟩ into a common edge, also ⟨*starring*⟩ and ⟨*occupation*⟩ in the same way. Note that we cannot make edges with ⟨*type*⟩ as a combination, because they don't share the same prefix. And we mine frequent patterns of these four Compressed Graphs. {P_1,P_2,P_3,P_4,P_5} is a subset of P.

Candidate-selection: Reconstitute candidates in P with labels of entities. By following steps, we can obtain the refined query $\mathcal{Q}'$:

(1) *Calculate Weight* The weight of each graph P_i in P comprises 4 parts: the number of answers covered in $\mathcal{R}$, the edit distance between P_i and original query Q, the number of extra answers introduced by P_i (never show in $Q(D)$ and $\mathcal{R}$), the number of answers in $Q^-(D)$ as punishment. Each part is multiplied by a parameter, determining the significance of either precision or similarity.

(2) *Weighted Set Cover* For all members in P, each with a weight, use a greedy algorithm to find a subset $\mathcal{Q}'$ of P, which can cover all the answers in $\mathcal{R}$, aiming at acquiring minimize weight. Subset $\mathcal{Q}'$ corresponds to the refined SPARQL.

See Figure 2, P_5 covers $\{r_2, r_3, r_4\}$. It's the most similar to Q because only ⟨*deathPlace*⟩ is changed into ⟨*birthPlace*⟩. However, it brings in 34 extra answers and 3 known wrong answers. So it has a large weight. Finally, we choose $\{P_2, P_3\}$ as the best cover.

Log Dealing *Rule Mining:* Once modify a graph successfully, analyze the modification log and extract the rules. Discover new knowledge to correct and supplement existing entity-mapping dictionary and relation-paraphrase dictionary, which will avoid error (1) and (2) for future queries (see Figure 1). Also build up a sentence-structure dictionary to record query structures corresponding to specific natural language question(e.g., "be in country" has two corresponding templates:"in . . . country" and "in a city of . . . country"), which will avert error (3) for subsequent queries. In such way we build a mechanism to improve subsequent queries by self-learning.

3. EXPERIMENTS

Our experiment is implemented on a 7.8GB DBpedia dataset. We choose 100 natural questions from QALD-5 whose original answers are not perfect generated by an existing system gAnswer described in [2]. Our algorithm can modify 42 of them and receive correct SPARQL, which solves the three typical errors we proposed before. Among those 42 cases which are solved perfectly, 84% of them are fixed in no more than 2 options. For other 58 cases, the precision improves 30%, recall increases around 80%. After these modifications, entity-mapping dictionary and relation-paraphrase dictionary are rectified and sentence-structure dictionary is built up. When using appeared questions to query the system, the system never shows similar errors and receives perfect answers.

Acknowledgments. This work was supported by The National Key Research and Development Program of China under grant 2016YFB1000603 and NSFC under grant 61622201, 61532010, 61370055 and 61402020. Lei Zou is the corresponding author of this paper.

4. REFERENCES

[1] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *ICDM*, 2002.

[2] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over rdf: a graph data driven approach. In *SIGMOD*, 2014.