Knowl Inf Syst
https://doi.org/10.1007/s10115-018-1155-4

REGULAR PAPER

Answering why-not questions on SPARQL queries

Meng Wang[1] · Jun Liu[1] · Bifan Wei[1] · Siyu Yao[1] · Hongwei Zeng[1] · Lei Shi[1]

Received: 14 March 2017 / Revised: 18 November 2017 / Accepted: 5 January 2018

Abstract SPARQL, the W3C standard for RDF query languages, has gained significant popularity in recent years. An increasing amount of effort is currently being exerted to improve the functionality and usability of SPARQL-based search engines. However, explaining missing items in the results of SPARQL queries or the so-called why-not question has not received sufficient attention. In this study, we first formalize why-not questions on SPARQL queries and then propose a novel explanation model, called answering why-not questions on SPARQL (ANNA) to answer why-not questions using a divide-and-conquer strategy. ANNA adopts a graph-based approach and an operator-based approach to generate logical explanations at the triple pattern level and the query operator level, respectively, which helps users refine their initial queries. Extensive experimental results on two real-world RDF datasets show that the proposed model and algorithms can provide high-quality explanations in terms of both effectiveness and efficiency.

Keywords Why-not · SPARQL · RDF graph · Query · Graph pattern

✉ Meng Wang
wangmengsd@stu.xjtu.edu.cn

Jun Liu
liukeen@xjtu.edu.cn

Bifan Wei
weibifan@mail.xjtu.edu.cn

Siyu Yao
cheryl@stu.xjtu.edu.cn

Hongwei Zeng
zhw1025@gmail.com

Lei Shi
xjtushilei@foxmail.com

[1] MOEKLINNS Lab, Xi'an Jiaotong University, Xi'an, China

Published online: 19 January 2018

1 Introduction

Users always have expectations for query results. They feel frustrated when the result is empty or the result does not contain the expected items. Hence, a user would naturally pose a why-not question on why some items do not show up in the result set. Answering why-not questions helps users clarify their information needs and refine their initial queries.

Existing efforts to answer why-not questions mainly focus on the area of relational queries [3,7,8,16,18–20,23,33]. However, no investigation has been conducted on RDF datasets.

1.1 Why-not questions on SPARQL queries

Querying large collections of RDF datasets, such as Linked Data,[1] has gained significant popularity in recent years. SPARQL [17] has become the de facto standard query language in this domain. Why-not questions also occur during SPARQL querying, as illustrated in Example 1.

Example 1 A user wants to find all films directed by *Tim Burton* since *1990*. After posing a SPARQL query Q_1 over DBpedia,[2] the user finds that the famous film *Batman* is not in the result set, as shown in Fig. 1. Consequently, the user may issue a why-not question, i.e., why *Batman* does not appear in the result set.

The answer to this why-not question could be that the film is not in DBpedia, the film is not directed by *Tim Burton*, the film is released before 1990, or a bug exists in the SPARQL query processing engine. Worse, even if the user asks another similar question, e.g., why *The Nightmare Before Christmas* does not appear in the result set, the reason may be different from the former.[3]

Faced with such why-not questions, users have no idea which parts of the query should be responsible for the missing items. Users experience difficulty sifting their initial SPARQL queries. We refer to this problem as answering why-not questions on SPARQL queries. In this study, we focus on this problem.

1.2 Limitations of existing explanation models

Existing explanation models that answer why-not questions fall into three categories, namely instance based [19,20], operator based [3,7] and query refinement based [8,16,18,23,33].

Instance-based models Instance-based models focus on the data in relational databases [19,20], which illustrate how the data source should be updated if users want missing items to appear in the result set. For example, in our *missing-Batman* example, by modifying *Batman*'s release date *1989* to an arbitrary value satisfying *value* $>$ 1990 or changing *Big Fish*'s film name in the database to *Batman*, *Batman* will appear in the result set.

RDF datasets have no definite schema or centralized data design; therefore, computing all instance-based explanations may be impossible for instance-based models.

Operator-based models Operator-based models generate explanations for why-not questions by identifying which query operators eliminate the missing items from the result set [3,7]. These models also focus on relational queries, such as Select-Project-Join-Union-Aggregation (SPJUA) queries [3], which are based on relational algebra.

[1] http://linkeddata.org.

[2] http://wiki.dbpedia.org.

[3] *Batman* was released in 1989. *The Nightmare Before Christmas* was written and produced by *Tim Burton*, but its director was *Henry Selick*.

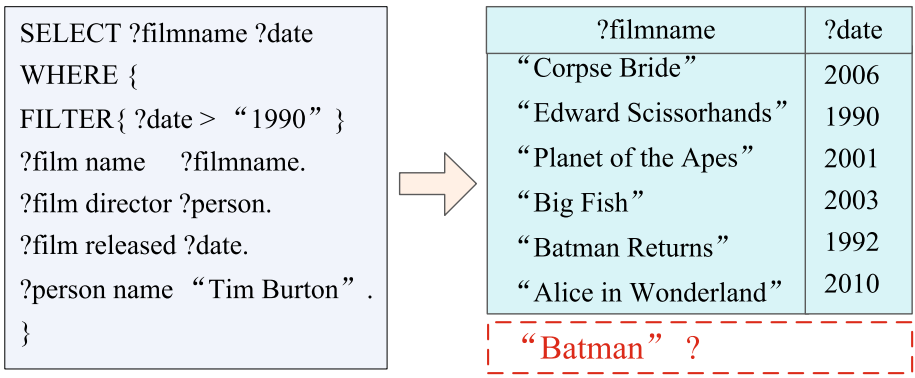


Fig. 1 SPARQL query and result set for Example 1. **a** SPARQL query Q_1. **b** Query result

SPARQL query operators are based on SPARQL algebra [28]. The fundamental differences between SPARQL algebra and relational algebra make operator-based models unsuitable for SPARQL queries. Firstly, the notion of so-called mappings [28] is central to the evaluation process of SPARQL instead of tuples in the SQL evaluation process [3]. Secondly, if the why-not question is caused by inappropriate parts of the basic graph patterns (BGPs) of the SPARQL query, operator-based models would always output *JOIN* as the explanation, and they cannot tell users which parts of BGPs are responsible for excluding expected items.

Query refinement-based models Query refinement-based models tell users how to refine their original queries so that the missing items can return to the result. Nonetheless, why-not questions are query dependent [16], and a query refinement-based model is only applicable to a certain class of queries.

For instance, the model in [33] focuses on refining the constraints in SPJUA queries to recover missing items. For top-k queries, the model in [18] makes the missing items appear in the result set by changing the value k of the top-k query or users' preference weights on attributes of the dataset. Other existing works can answer why-not questions on reverse top-k queries [16], spatial keyword top-k queries [8] and reverse skyline queries [23]. None of the existing models can be applied directly to SPARQL queries.

1.3 Overview of ANNA

Answering why-not questions on SPARQL queries has the following three main challenges:

Why-not reasons RDF graphs are labeled directed graphs and SPARQL is essentially a graph-matching query language [28]. The evaluation of a SPARQL query mainly includes (1) matching graph patterns against the input RDF graph and (2) applying SPARQL algebraic operations in the matching process. Determining the parts that are responsible for excluding users' expected items is challenging. It necessitates understanding the semantics of the why-not questions on SPARQL queries to analyze underlying reasons and to compute what is necessary for the missing items to appear in the result.

Effectiveness We need to guarantee that explanations are effective. One straightforward method is employing query relaxation models to recover the missing items and to return the relaxed queries as explanations. However, existing relaxation models only cover the

conjunctive queries and do not consider inappropriate directions of edges in SPARQL graph patterns.

Efficiency Finding inappropriate parts in the initial SPARQL query needs an examination of the matching process between the graph pattern and the input RDF graph in a linear fashion way [34]. This process may lead to an exhaustive search [30]. Therefore, the efficient computation of why-not explanations is challenging and non-trivial.

To address the above challenges, we designed a novel explanation model called ANNA[4] (**A**nswering why-**N**ot questio**N**s on SP**A**RQL). The procedure of ANNA mainly includes the following:

(i) ANNA computes a basic graph pattern from the SPARQL query, which is for the matching against the input RDF graph and is a necessity for missing items to appear, then ANNA examines the result of graph pattern matching and identifies the query level where the absence of the expected items occurs, i.e., the triple pattern level or the query operator level.
(ii) If the absence occurs at the triple pattern level, ANNA will detect both inappropriate directions and RDF terms by using a graph-based approach and generate a modified graph pattern. The expected items will be returned to the query result by matching the modified graph pattern against the input RDF graph. Then, the modified graph pattern will be utilized as an explanation.
(iii) If the absence occurs at the query operator level, ANNA will trace the expected items on the query parse tree through a post-order traversal. To save computation time, only questionable operators will be examined during the traversal. By comparing expected items with the result of each questionable operator, ANNA will detect inappropriate operators that filter out expected items and return the operator and the corresponding conditions to users as an explanation.

1.4 Contributions and organization

The main contributions of this study are threefold.

(i) We first formalize why-not questions on SPARQL queries and reveal their semantics.
(ii) We propose a novel explanation model, ANNA, to answer why-not questions on SPARQL queries. We propose algorithms for ANNA to generate explanations at different query levels using a divide-and-conquer strategy.
(iii) We conducted extensive experiments on two real-world RDF datasets (LinkedMDB[5] and DBpedia[6]) to confirm the effectiveness and efficiency of ANNA. Experimental results show that effective explanations for why-not questions can be generated within an acceptable time at both triple pattern and query operator levels.

The remainder of this paper is organized as follows: Related work is discussed in Sect. 2. The why-not question on SPARQL queries is formalized in Sect. 3, and the underlying reasons are analyzed in Sect. 4. Section 5 presents the framework of the proposed model, and the details about how explanations are generated. The evaluation of the model is reported in Sect. 6. Finally, the conclusions and the future work are presented in Sect. 7.

[4] A demo system of ANNA has been presented at *ISWC 2015*.

[5] http://queens.db.toronto.edu/~oktie/linkedmdb/.

[6] http://wiki.dbpedia.org/Datasets, released in September, 2015.

2 Related work

To the best of our knowledge, no model has addressed why-not questions on SPARQL queries. We briefly discuss the research efforts related to our work in the following aspects: explanation models for why-not questions, query relaxation for RDF data, inconsistency-tolerant query answering and provenance for SPARQL queries.

2.1 Explanation models for why-not questions

As introduced in Sect. 1.2, three types of models can be used to answer why-not questions: **instance-based models** [19,20], **operator-based models** [3,7] and **query refinement-based explanation models** [8,16,18,23,33].

Recently, answering why-not questions has also received attention in other research fields. Bhowmick et al. [2] and Wang et al. [35] designed query refinement-based models to answer why-not questions in image search. Cate et al. [31] introduced an ontology-based model for explaining why-not questions on conjunctive queries. Calvanese et al. [6] and Bienvenu et al. [4] leveraged abductive reasoning to answer why-not questions on the data represented by a DL-Lite ontology. For graph databases, Saiful et al. [24] proposed a query refinement-based model to address the problem of answering why-not questions on similar graph matching. However, this model concerns simple undirected graphs, whereas RDF graphs are labeled directed graphs. Moreover, the model in [24] can only answer why-not questions caused by inappropriate topological structures of query graphs, whereas both inappropriate RDF terms and algebraic operations in SPARQL queries can exclude the expected items of users.

2.2 Inconsistency-tolerant query answering

In description logic (DL) knowledge bases field, it is well known that the inconsistency of instances part (ABox) leads to missing answers in the result set [26]. Inconsistency-tolerant semantics [1,5,12,26,37] aim to provide meaningful answers to the queries even when the data conflict with the ontology (TBox). The most well known is the brave [5], AR [26] and IAR semantics [26]. Recently, a framework was proposed in [4] for explaining positive and negative query answers (similar to why-not questions) under inconsistency-tolerant semantics. However, the model in [4] is only applicable to explain the missing answers caused by the inconsistent subset of ABox, and it is a data-centric approach which focus on finding the original conflict assertions (responsible for eliminating answers) in ABox. In contrast, even with the consistent knowledge base, why-not questions on SPARQL queries may be caused by inappropriate queries, and the goal of our paper is to design a model to modify the original query to make the missing answers appear and help users refine their initial queries.

2.3 Query relaxation for RDF data

Query relaxation for RDF data has been studied to address the empty/few-answers problem: the user's query is too selective, and the number of answers is not sufficient [22], which is different from the why-not problem. Query relaxation models attempt to reformulate the original query into a new relaxed query by removing or relaxing conditions so that the result of the new query contains sufficient answers. Two types of models (i.e., similarity-based and rule-based models) are currently utilized to generate multiple relaxed query candidates. Similarity-based models [13,14] leverage lexical analyses to determine appropriate relaxation

candidates. Rule-based models [21,22,25,38] exploit RDF schema (RDFS) semantics to perform relaxation.

2.4 Provenance for SPARQL queries

Data provenance has been studied to help us understand why the items exist in the result set [9]. For SPARQL queries, existing works [10,11,32] focus on explaining the provenance of RDF data to evaluate the data quality and trustworthiness. Data provenance can be used to answer why questions (i.e., why a piece of data is in the query result). However, it cannot be applied to our problem as answering why-not questions that concern the items that do not appear in the result set.

3 Problem formulation

Before defining the why-not question on SPARQL queries, we follow the official W3C SPARQL standard [17] and the existing formalization of SPARQL in [28] to briefly introduce several notations employed in this study.

3.1 Preliminaries

Definition 1 (*RDF graph*) Let I be the set of Internationalized Resource Identifiers (IRIs), L be the set of literals (denoted by quoted strings, e.g., "Tim Burton″) and B be the set of blank nodes. An RDF term is a member of the set $T = I \cup L \cup B$. An RDF triple $t = \langle subject, predicate, object \rangle$ is a member of the set $(I \cup B) \times I \times T$. An RDF graph is a finite set of RDF triples. In this study, we also refer to an RDF graph as an RDF dataset.

Definition 2 (*Graph pattern*) [28] Let V be a set of query variables, which is disjoint from T and is distinguished by leading question mark symbols, e.g., ?filmname, ?person. SPARQL graph patterns are defined recursively as follows:

(1) A tuple $tp \in (I \cup L \cup V) \times (I \cup V) \times (T \cup V)$ that contains variables in the subject, predicate or object is a graph pattern. A tp is also called a triple pattern.
(2) If P_1 and P_2 are graph patterns, then P_1 *AND* P_2, P_1 *UNION* P_2, P_1 *MINUS* P_2 and P_1 *OPTIONAL* P_2 are graph patterns, where *AND*, *UNION*, *MINUS* and *OPTIONAL* are SPARQL operators [17].
(3) If P is a graph pattern, then P *FILTER* R is a graph pattern, where *FILTER* is a SPARQL operator and R is a filter condition. A filter condition is a unary or binary expression [17], such as $?date >$ "1990."

A graph pattern P is called a basic graph pattern (BGP) if it only concatenates triple patterns by *AND*. A BGP P can be denoted by a set form, such as $P = \{tp_1, tp_2, \ldots, tp_n\}$. A triple pattern can also be considered as a BGP. BGPs are the basic building blocks for any other graph patterns.

Definition 3 (*SPARQL query*) The official standard [17] defines four different forms of queries on the top of graph patterns, namely *SELECT*, *ASK*, *CONSTRUCT* and *DESCRIBE*. We will restrict our discussion to *SELECT*, which is the only form that can return the graph-matching results to users. We further define a SPARQL query Q as an expression of the form *SELECT* S *WHERE* P, where P is a graph pattern and $S \subseteq var(P)$. We use $var(P)$ to denote the set of variables that occur in the graph pattern P.

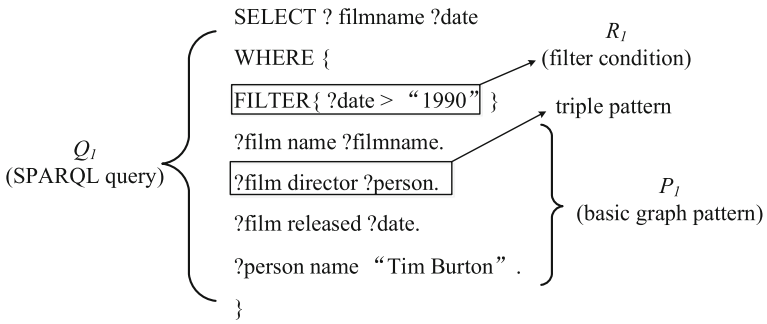


Fig. 2 SPARQL query Q_1 for Example 1

The SPARQL query Q_1 for Example 1 is annotated in Fig. 2 to better understand *SPARQL query, filter condition, triple pattern* and *basic graph pattern*.

Definition 4 (*Mapping*) A mapping is a **partial function** $\mu : V \rightarrow T$. For a triple pattern tp, we use $\mu(tp)$ to denote the triple obtained by replacing the $var(tp)$ (variables in tp) with RDF terms according to μ. The domain $dom(\mu)$ of a mapping μ is the set of variables on which μ is defined. We say that two mappings μ_1, μ_2 are compatible, denoted by $\mu_1 \sim \mu_2$, if $\mu_1(?x) = \mu_2(?x)$ for all $?x \in dom(\mu_1) \cap dom(\mu_2)$.

For instance, assuming that $\mu_1 = \{?film \rightarrow f1336, ?filmname \rightarrow$ "*BigFish*"$\}$, $\mu_2 = \{?film \rightarrow f1336, ?date \rightarrow$ "2003"$\}$, $dom(\mu_1) = \{?film, ?filmname\}$ and $dom(\mu_2) = \{?film, ?date\}$, then $\mu_1 \sim \mu_2$ because of $\mu_1(?film)$=$\mu_2(?film)$=$f1336$ and $?film \in dom(\mu_1) \cap dom(\mu_2)$.

Given a mapping μ and a filter condition R, we use $\mu \models R$ to present that μ satisfies R. For example, if R is $?date >$ "1990," $?date \in dom(\mu)$ and $\mu(?date) =$ "1995," then $\mu \models R$ because of "1995" > "1990."

Definition 5 (*SPARQL algebraic operation*) Given two sets of mappings Ω_1 and Ω_2, R denotes a filter condition, and $Y \subseteq V$ is a finite set of variables, the SPARQL algebraic operations join ($\bowtie$), union ($\cup$), difference ($\setminus$), left outer join ($⟕$), projection (π) and selection (σ) are defined as follows:

$$
\begin{aligned}
&\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 : \mu_1 \sim \mu_2\},\\
&\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\},\\
&\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2 : \mu_1 \nsim \mu_2\},\\
&\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2),\\
&\pi_Y(\Omega) = \{\mu \mid \exists \mu', \mu \cup \mu' \in \Omega \ \wedge \ dom(\mu) \subseteq Y \wedge\\
&\qquad\qquad dom(\mu') \cap Y = \varnothing\},\\
&\sigma_R(\Omega) = \{\mu \in \Omega \mid \mu \models R\}.
\end{aligned}
\tag{1}
$$

Definition 6 (*SPARQL semantic*) Let D be an RDF dataset, tp a triple pattern, P, P_1, P_2 graph patterns, R a filter condition and Q (*SELECT S WHERE P*) a SPARQL query. We define the semantics of graph patterns and Q as follows:

$$
\begin{aligned}
&[\![tp]\!]_D = \{\mu | dom(\mu) = var(tp) \text{ and } \mu(tp) \in D\},\\
&[\![P_1\ AND\ P_2]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D,\\
&[\![P_1\ OPTIONAL\ P_2]\!]_D = [\![P_1]\!]_D \rtimes [\![P_2]\!]_D,\\
&[\![P_1\ UNION\ P_2]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D,\\
&[\![P_1\ MINUS\ P_2]\!]_D = [\![P_1]\!]_D \setminus [\![P_2]\!]_D,\\
&[\![P\ FILTER\ R]\!]_D = \sigma_R([\![P]\!]_D),\\
&[\![Q]\!]_D = \pi_S([\![P]\!]_D),
\end{aligned}
\tag{2}
$$

where $[\![.]\!]_D$ is a function that takes a graph pattern or a SPARQL query to match against the RDF dataset D and returns a set of mappings. In the following part of this study, we use SPARQL algebraic operations and operators interchangeably, e.g., $P_1\ AND\ P_2$ can be denoted as $P_1 \bowtie P_2$.

The following example illustrates *mapping, SPARQL algebraic operation* and *SPARQL semantic*.

Example 2 Consider the SPARQL query Q_2

```
SELECT ?filmname ?date
  WHERE {
    ?film  name  ?filmname.
    ?film  director  ?person.
    ?person  name  "Tim Burton"
    OPTIONAL{ ?film  released  ?date}
  },
```

which retrieves all *Tim Burton*'s film (*?filmname*) and optionally (i.e., if available), their release date (*?date*). The graph pattern of Q_2 is $P_1 \rtimes P_2$, where

$P_1 = \{\langle$*?film name ?filmname*$\rangle$.
$\langle$*?film director ?person*$\rangle$.
$\langle$*?person name* "*Tim Burton*"$\rangle\}$,
$P_2 = \{\langle$*?film released ?date*$\rangle\}$,

$\rtimes$ (*OPTIONAL*) is the algebraic operation.

Figure 3 illustrates an RDF graph D_2 in Example 2, *f1342*, *f1336*, *f1333* and *p2556* are IRIs.

The following is the semantic of $P_1 \rtimes P_2$ on D_2:

$$
\begin{aligned}
&[\![P_1 \rtimes P_2]\!]_{D_2} =\\
&\{\{?film \rightarrow f1336, ?filmname \rightarrow \text{“}BigFish,\text{”}\\
&\qquad \underbrace{?person \rightarrow p2556, ?date \rightarrow \text{“}2003\text{”}\}}_{\mu_1},\\
&\{?film \rightarrow f1342, ?filmname \rightarrow \text{“}CorpseBride,\text{”}\\
&\qquad \underbrace{?person \rightarrow p2556, ?date \rightarrow \text{“}2005\text{”}\}}_{\mu_2},\\
&\{?film \rightarrow f1333, ?filmname \rightarrow \text{“}Batman,\text{”}\\
&\qquad \underbrace{?person \rightarrow p2556\}}_{\mu_3}\},
\end{aligned}
$$

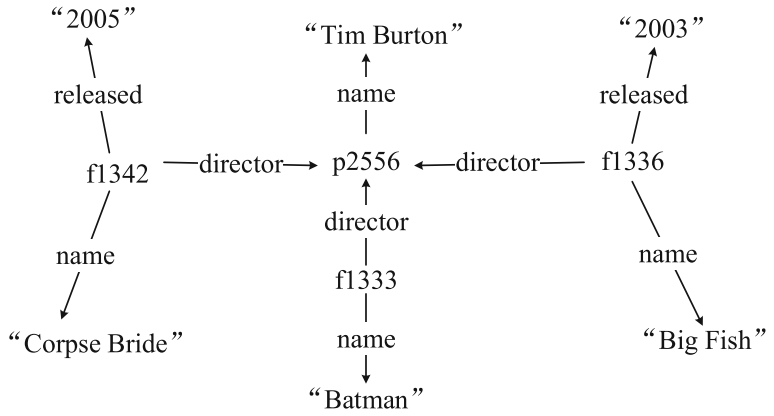


Fig. 3 RDF graph D_2 in Example 2

where μ_1, μ_2 and μ_3 are mappings. The following can be verified easily:

$$\begin{aligned}[\![Q_2]\!]_{D_2} = \{&\{?filmname \rightarrow \text{“}BigFish,\text{”} ?date \rightarrow \text{“2003”}\},\\ &\{?filmname \rightarrow \text{“}CorpseBride,\text{”} ?date \rightarrow \text{“2005”}\},\\ &\{?filmname \rightarrow \text{“}Batman\text{”}\}\}.\end{aligned}$$

Note that *OPTIONAL* allows available information to be added to a mapping μ. If the optional part of the graph pattern does not match the data, then the relevant variables are left unbound.

3.2 SPARQL why-not questions

Definition 7 (*Why-not question*) Given a SPARQL query Q (*SELECT S WHERE P*) on D and its result set $[\![Q]\!]_D$, a why-not question on Q is defined as an expected mapping μ_w posed by users, where $dom(\mu_w) \subseteq S$ and $\forall \mu \in [\![Q]\!]_D : \mu_w \nsubseteq \mu$.

Consider the SPARQL query Q_1 in Example 1, a user may ask why a 1995 film, *Batman Forever*, is missing in query results. This why-not question can be denoted by mapping {?*filmname* → “*Batman Forever*, ”?*date* → “1995”}.

Definition 8 (*Explanation*) Given a why-not question μ_w on Q, an explanation ψ represents the reason for μ_w. Two forms of explanation exist according to the different why-not reasons (a detailed analysis is given in Sect. 4):

(1) A modified graph pattern P', which is similar to the original P of Q. We say $\psi = P'$ if $\exists \mu' \in [\![P']\!]_D : \mu_w \subseteq \mu'$. Consider the why-not question $\mu_w = \{?filmname \rightarrow \text{“}Batman\text{”}\}$ in Example 1, explanation ψ may be a modified P', where

$$\begin{aligned}P' = \{&\langle ?\textit{film name} ?\textit{filmname}\rangle.\\ &\langle ?\textit{film}\ \textbf{\textit{producer}}\ ?\textit{person}\rangle.\\ &\langle ?\textit{film released}\ ?\textit{date}\rangle.\\ &\langle ?\textit{person name}\ \text{“Tim Burton”}\rangle\}.\end{aligned}$$

$\psi = P'$ indicates that the *director* of *Batman* is not *Tim Burton*.

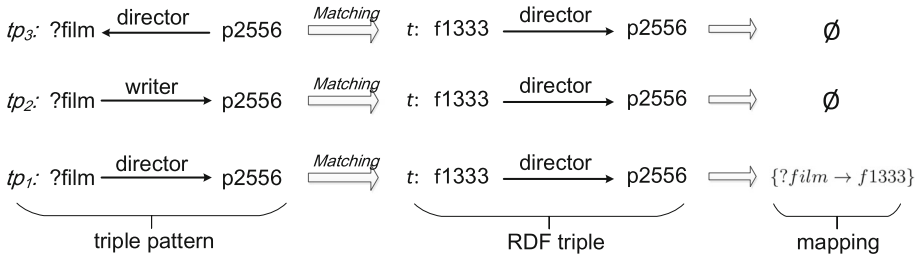


Fig. 4 Evaluations of ⟨*?film director p2556*⟩ on RDF triples t_1, t_2, and t_3, respectively

(2) A set of tuples, denoted by $\{(op_i, \mu_i)\}_l$, where each tuple (op_i, μ_i) indicates a query operator op_i and the corresponding mapping $\mu_i \in [\![P]\!]_D$ that satisfies $\mu_w \subseteq \mu_i$. Following the example of $\mu_w = \{?filmname \rightarrow \text{“}Batman''\}$, the explanation ψ may also be a set

$$\{(\text{FILTER}(?date > \text{“}1990''),\ \{?film \rightarrow f1333,\ ?date \rightarrow \text{“}1989,''\ ?person \rightarrow p2556,\ ?filmname \rightarrow \text{“}Batman''\}\}.$$

The set indicates that *Batman* is filtered out because of the filter condition $?date >$ “1990.″

4 Problem analysis

Our investigation found that inappropriate graph patterns, incompleteness/inconsistency of the RDF database and spelling/syntactic errors of SPARQL queries may lead to why-not questions on SPARQL queries. In this study, we suppose that the RDF database is clean and consistent with ontologies, and the SPARQL query of users is free of spelling/syntactic errors. We focus on why-not questions caused by inappropriate graph patterns, and this already yields a non-trivial framework to study.

Given a SPARQL query Q over D, the evaluation $[\![P]\!]_D$ of Q can be divided into two levels [17]: the triple pattern level and the query operator level. At the triple pattern level, each triple pattern of P is matched against D to obtain a set of mappings. Then, SPARQL algebraic operations ($\bowtie$, $\cup$, $\setminus$, $\rtimes$, π) are applied on the mappings to produce the query answer at the query operator level.

4.1 “Why-Not” at the triple pattern level

Given a triple pattern tp and a why-not question μ_w, $\mu_w \notin [\![tp]\!]_D$ may be caused by **an inappropriate direction** or **inappropriate RDF terms** in tp. Example 3 illustrates the different scenarios.

Example 3 Considering an RDF triple $t = \langle f1333\ director\ p2556\rangle$, tp_1, tp_2 and tp_3 are three possible triple patterns to be evaluated on t, respectively, as shown in Fig. 4. The evaluation $[\![tp_1]\!]_t = \{?film \rightarrow f1333\}$ will not yield a missing answer. In the cases of tp_2 and tp_3, the why-not question $\mu_w = \{?film \rightarrow f1333\}$ will be proposed to $[\![tp_2]\!]_t$, $[\![tp_3]\!]_t$, respectively.

For $[\![tp_2]\!]_t = \varnothing$ and $\mu_w = \{?film \rightarrow f1333\}$, if the RDF term *writer* of tp_2 is modified to *director*, then $[\![tp_2{}']\!]_t = \{?film \rightarrow f1333\}$. The reason for μ_w is that **inappropriate**

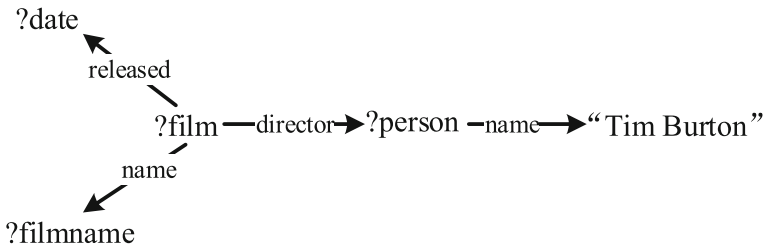


Fig. 5 BGP in EXAMPLE 1

RDF terms in tp_2 led to a failure to match t. We define a function $Modify()$ that modifies a triple pattern tp by substituting an RDF term with another one in the RDF dataset or a variable, and $\mu_w \subseteq [\![Modify(tp)]\!]_D$.

For $[\![tp_3]\!]_t = \varnothing$, the reason for $\mu_w = \{?film \rightarrow f1333\}$ is that the triple pattern formulated by the user is in an inappropriate direction. If we modify tp_3 to $tp_3' = \langle ?film\ director\ p2556\rangle$ just by switching the direction of *director* in tp_3, then tp_3' will match t and $[\![tp_3{}']\!]_t = \{?film \rightarrow f1333\}$. Formally, we define a function $Reverse()$ that modifies a triple pattern tp to tp' by reversing the direction of the predicate in tp. Then, for a why-not question $\mu_w \nsubseteq [\![tp]\!]_D$, tp is in **an inappropriate direction** if and only if $\mu_w \subseteq [\![Reverse(tp)]\!]_D$.

4.2 "Why-Not" at the query operator level

Given an RDF graph D and a graph pattern P, μ_w is a why-not question on P, i.e., $dom(\mu_w) \subseteq var(P)$ and $\forall \mu \in [\![P]\!]_D : \mu_w \nsubseteq \mu$.

We first analyze μ_w on P with the same kind of operators, and then, we proceed to perform a higher-level analysis on the combination of different operators.

AND Let $P = P_1 \bowtie P_2 \ldots \bowtie P_n$, where P_1, P_2, ..., P_n are BGPs. P is also a BGP (cf. Definition 2). Triple patterns in P concatenate each other by $\bowtie$ and form a connected graph. Figure 5 shows a graphical representation of the BGP in Example 1.

Each mapping in $[\![P]\!]_D$ presents a match of P, which is a RDF subgraph of D. The reason for μ_w on P is that some triple patterns in P are inappropriate to match the RDF subgraph of D about μ_w. Adapting the idea from explaining why-not questions in Sect. 4.1, the explanation ψ_P is a modified graph pattern P', which satisfies $\exists \mu' \in [\![P']\!]_D : \mu_w \subseteq \mu'$. Note that if $[\![P]\!]_D \neq \varnothing$, then P matches at least one RDF subgraph in D and all directions of predicates in P are correct. Modifying inappropriate triple patterns should focus on inappropriate RDF terms. If $[\![P]\!]_D = \varnothing$, then both directions and RDF terms should be considered for modifications.

UNION Let $P = P_1 \cup P_2 \cdots \cup P_n$, where P_1, P_2, ..., P_n are BGPs. Applying $[\![P_1 \cup P_2]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$ (cf. Definition 6) to $[\![P]\!]_D$ recursively, we can obtain $[\![P]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D \cdots \cup [\![P_n]\!]_D = \{\mu \mid \mu \in [\![P_1]\!]_D$ or $\mu \in [\![P_2]\!]_D \ \ldots$ or $\mu \in [\![P_n]\!]_D\}$. With the premise $\forall \mu \in [\![P]\!]_D : \mu_w \nsubseteq \mu$ (cf. Definition 7), we can obtain $\forall \mu \in [\![P_i]\!]_D : \mu_w \nsubseteq \mu$, where $i = 1, 2, \ldots, n$. It is intuitive that the explanation for the why-not question μ_w on P is a set of explanations $\Psi_P = \{\psi_{P_1}, \psi_{P_2}, \ldots, \psi_{P_n}\}$, where $\psi_{P_1}, \psi_{P_2}, \ldots, \psi_{P_n}$ are explanations for μ_w on P_1, P_2, ..., P_n, respectively.

Example 4 Consider a graph pattern $P = P_1 \cup P_2$ is used to find *war or road films released in America*, where

$$P_1 = \{\langle ?film\ name\ ?filmname\rangle.$$

$$\langle ?film\ country\ \text{“}United\ States\text{”}\rangle.$$
$$\langle ?film\ type\ War\ films\rangle\},$$
$$P_2 = \{\langle ?film\ name\ ?filmname\rangle.$$
$$\langle ?film\ country\ \text{“}United\ States\text{”}\rangle.$$
$$\langle ?film\ type\ Road\ films\rangle\}.$$

After graph pattern matching, we may obtain following mappings for P.

$[\![P_1]\!]_D = \{\{?film \rightarrow f1137, ?filmname \rightarrow \text{“}Fury\text{”}\},$
$\{?film \rightarrow f2209, ?filmname \rightarrow \text{“}Saving\ Private\ Ryan\text{”}\}\ \}.$
$[\![P_2]\!]_D = \{\{?film \rightarrow f5953, ?filmname \rightarrow \text{“}Rain\ Man\text{”}\},$
$\{?film \rightarrow 21549, ?filmname \rightarrow \text{“}Nebraska\text{”}\}\}.$
$[\![P]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$
$= \{\{?film \rightarrow f1137, ?filmname \rightarrow \text{“}Fury\text{”}\},$
$\{?film \rightarrow f5953, ?filmname \rightarrow \text{“}Rain\ Man\text{”}\},$
$\{?film \rightarrow 21549, ?filmname \rightarrow \text{“}Nebraska\text{”}\},$
$\{?film \rightarrow f2209, ?filmname \rightarrow \text{“}Saving\ Private\ Ryan\text{”}\}\ \}.$

A why-not question $\mu_w = \{?filmname \rightarrow \text{“}Atonement\text{”}\}$ is posed on P. The explanation for $\mu_w = \{?filmname \rightarrow \text{“}Atonement\text{”}\}$ on P is $\Psi_P = \{\psi_{P_1}, \psi_{P_2}\}$, where

$$\psi_{P_1} = \{\langle ?\ film\ name\ ?filmname\rangle.$$
$$\langle ?\ film\ country\ \text{“}United\ Kingdom\text{”}\rangle.$$
$$\langle ?film\ type\ War\ films\rangle\},$$
$$\psi_{P_2} = \{\langle ?film\ name\ ?filmname\rangle.$$
$$\langle ?film\ country\ \text{“}United\ Kingdom\text{”}\rangle.$$
$$\langle ?film\ type\ Romantic\ films\rangle\}.$$

For μ_w on P with ***MINUS***, ***OPTIONAL*** and ***FILTER***, we propose **Propositions** 1-3 to simplify μ_w on ***AND*** and ***UNION*** explanations.

Proposition 1 *Let $P = P_1 \setminus P_2$, where P_1, P_2 are BGPs, the explanation for the why-not question μ_w on P is denoted by ψ_P.*

(1) *If $\exists \mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$, then $(MINUS(\mu_2), \mu_1) \in \psi_P$, which indicates that μ_1 is eliminated from result mappings by the operator $MINUS$ due to $\mu_2 \in [\![P_2]\!]_D$ and $\mu_1 \sim \mu_2$.*
(2) *If $\forall \mu_1 \in [\![P_1]\!]_D : \mu_w \nsubseteq \mu_1$, then $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.*

Example 5 Consider a graph pattern $P = P_1 \setminus P_2$ is used to find *war films not released in America*, where

$$P_1 = \{\langle ?film\ name\ ?filmname\rangle.$$
$$\langle ?film\ type\ War\ films\rangle\},$$
$$P_2 = \{\langle ?film\ country\ \text{“}United\ States\text{”}\rangle\}.$$

After graph pattern matching, we may obtain following mappings for P.

$$[\![P]\!]_D = \{\{?\ film \rightarrow f2356, ?filmname \rightarrow \text{“}The\ Pianist\text{”}\},$$
$$\{?\ film \rightarrow f3142, ?filmname \rightarrow \text{“}Hotel\ Rwanda\text{”}\}\}.$$

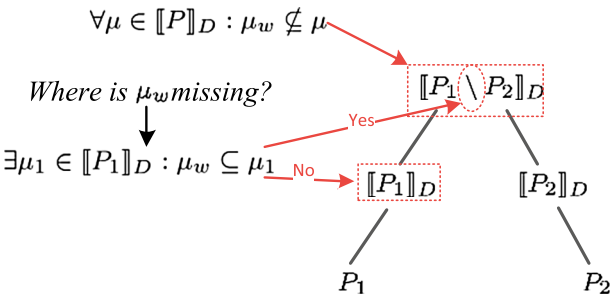


Fig. 6 The parser tree of $P = P_1 \setminus P_2$

A why-not question $\mu_w = \{?filmname \rightarrow$ "*Brave Heart*"$\}$ is posed on P.

According to Proposition 1, we should first verify whether $[\![P_1]\!]_D$ contains *Brave Heart* by verifying whether $\exists\mu_1 \in [\![P_1]\!]_D : \{?filmname \rightarrow$ "*Brave Heart*"$\} \subseteq \mu_1$. If $[\![P_1]\!]_D$ contains *Brave Heart*, explaining μ_w will focus on *MINUS*; otherwise, explaining μ_w will focus on P_1.

The following proof for Proposition 1 is illustrated with Example 5.

Proof Let $P = P_1 \setminus P_2$, where P_1, P_2 are BGPs.

(i) According to $[\![P_1 \setminus P_2]\!]_D = [\![P_1]\!]_D \setminus [\![P_2]\!]_D$ and $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall\mu_2 \in \Omega_2 : \mu_1 \nsim \mu_2\}$ (cf. Definitions 5, 6), $[\![P]\!]_D = [\![P_1]\!]_D \setminus [\![P_2]\!]_D = \{\mu_1 \in [\![P_1]\!]_D \mid \forall\mu_2 \in [\![P_2]\!]_D : \mu_1 \nsim \mu_2\}$.

(ii) Figure 6 shows the parser tree and query evaluation of $[\![P]\!]_D$. With the premise $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$ (cf. Definition 7) on the root node, we can observe that μ_w can be eliminated in two cases: eliminated by the *MINUS* operator or eliminated by the graph pattern matching of P_1. We can locate where μ_w is missing on the parser tree by determining whether $\exists\mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$.

(iii) If $\exists\mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$, $[\![P_1]\!]_D$ contains μ_w in the parser tree, and *MINUS* eliminates μ_1 from result mappings on the root of the parser tree, as shown in Fig. 6. Besides, μ_2 which satisfies $\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$ must exist in the right side of the root (cf. Definition 5). Hence, $(MINUS(\mu_2), \mu_1) \in \psi_P$, where $\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$.

Taking Example 5 to illustrate above step, we may obtain three mappings for P_1.

$$[\![P_1]\!]_D = \{\{?film \rightarrow f2356, ?filmname \rightarrow \text{"}The\ Pianist\text{"}\},\ \{?film \rightarrow f3142, ?filmname \rightarrow \text{"}Hotel\ Rwanda\text{"}\},\ \{?film \rightarrow f1589, ?filmname \rightarrow \text{"}Brave\ Heart\text{"}\}\}.$$

We can find that $\mu_w = \{?filmname \rightarrow$ "*Brave Heart*"$\}$ is contained by $\{?film \rightarrow f1589, ?filmname \rightarrow$ "*Brave Heart*"$\}$ in $[\![P_1]\!]_D$, the reason for missing μ_w is that *MINUS* eliminates the mapping $\{?film \rightarrow f1589, ?filmname \rightarrow$ "*Brave Heart*"$\}$ from the result set. There must be a mapping $\{?film \rightarrow f1589\}$ in $[\![P_2]\!]_D$ satisfying $\{?film \rightarrow f1589, ?filmname \rightarrow$ "*Brave Heart*"$\} \sim \{?film \rightarrow f1589\}$, which indicates that *Brave Heart is produced in America*. Therefore, (MINUS($\{?film \rightarrow f1589, ?film \rightarrow$ "*Brave Heart*"$\}$), $\{?film \rightarrow f1589\}$) will be added to ψ_P.

(iv) If $\forall\mu_1 \in [\![P_1]\!]_D : \mu_w \not\subseteq \mu_1$, as shown in Fig. 6, μ_w is missing in $[\![P_1]\!]_D$ of the parser tree. The graph pattern matching of P_1 eliminates μ_1 from result mappings. Hence, $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.

Taking Example 5 to illustrate above step, we may obtain two mappings for P_1.

$$[\![P_1]\!]_D = \{\{?film \rightarrow f2356, ?filmname \rightarrow \text{“}The\ Pianist''\},$$
$$\{?film \rightarrow f3142, ?filmname \rightarrow \text{“}Hotel\ Rwanda''\}\}.$$

Then, we can observe that none of $[\![P_1]\!]_D$ contains $\mu_w = \{?filmname \rightarrow \text{“}Brave\ Heart''\}$. Therefore, μ_w on $P = P_1 \setminus P_2$ can be simplified to μ_w on P and $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1. A possible explanation is

$$\psi_P = \psi_{P_1} = \{\langle ?film\ name\ ?filmname\rangle.$$
$$\langle ?film\ type\ film\rangle\}.$$

□

Proposition 2 *Let $P = P_1 \bowtie P_2$, where P_1, P_2 are BGPs, the explanation the why-not question for μ_w on P is denoted by ψ_P.*

(1) $[\![P]\!]_D = [\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$, but $\psi_P \neq \{\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}\}$, where $\psi_{P_1 \bowtie P_2}$, $\psi_{P_1 \setminus P_2}$ are explanations for μ_w on $P_1 \bowtie P_2$, $P_1 \setminus P_2$, respectively.

(2) If $dom(\mu_w) \cap var(P_2) = \varnothing$, then $\forall \mu \in [\![P_1]\!]_D : \mu_w \nsubseteq \mu$ and $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.

(3) If $dom(\mu_w) \cap var(P_2) \neq \varnothing$, then $\forall \mu \in [\![P_1 \bowtie P_2]\!]_D : \mu_w \nsubseteq \mu$ and $\psi_P = \psi_{P_1 \bowtie P_2}$, where $\psi_{P_1 \bowtie P_2}$ is the explanation for μ_w on $P_1 \bowtie P_2$.

Example 6 Consider a graph pattern $P = P_1 \bowtie P_2$ is used to retrieve *the actor and his country if he won Academy Male Best Actor*, where

$P_1 = \{\langle ?actor\ name\ ?actorname\rangle.$
$\langle ?actor\ type\ actor\rangle\},$
$P_2 = \{\langle ?actor\ country\ ?country\rangle.$
$\langle ?actor\ gender\ male\rangle.$
$\langle ?actor\ haswon\ \text{“}Academy\ Award\ for\ Best\ Actor''\rangle\}.$

After graph pattern matching, we may obtain following mappings for P.

$[\![P]\!]_D = \{\{?actor \rightarrow p1119, ?actorname \rightarrow \text{“}Brad\ Pitt''\},$
$\{?actor \rightarrow p1785, ?actorname \rightarrow \text{“}Leonardo\ DiCaprio''\},$
$\{?actor \rightarrow p1276, ?actorname \rightarrow \text{“}Matthew\ McConaughey,''$
$?country \rightarrow \text{“}United\ States''\},$
$\{?cactor \rightarrow p2313, ?actorname \rightarrow \text{“}Eddie\ Redmayne,''$
$?ccountry \rightarrow \text{“}United\ Kingdom''\}\}.$

If a why-not question $\mu_w = \{?actorname \rightarrow \text{“}Spike\ Lee''\}$ is posed on P, then $dom(\mu_w) \cap var(P_2) = \varnothing$. Explaining μ_w will focus on P_1 according to Proposition 2.

In another case, a why-not question $\mu'_w = \{?actorname \rightarrow \text{“}Leonardo\ DiCaprio,''$ $?country \rightarrow \text{“}United\ States''\}$ can be also posed on P, despite that $\{?actorname \rightarrow \text{“}Leonardo\ DiCaprio''\}$ is already in $[\![P]\!]_D$. According to Proposition 2, $dom(\mu'_w) \cap var(P_2) = \{?country\}$, explaining μ'_w is translated into explain μ'_w on $P_1 \bowtie P_2$.

The following proof for Proposition 2 is illustrated with Example 6.

Proof Let $P = P_1 \bowtie P_2$, where P_1, P_2 are BGPs.

(i) According to $[\![P_1 \bowtie P_2]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$ and $\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$ (cf. Definitions 5, 6), $[\![P]\!]_D = ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \cup ([\![P_1]\!]_D \setminus [\![P_2]\!]_D) = [\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$.

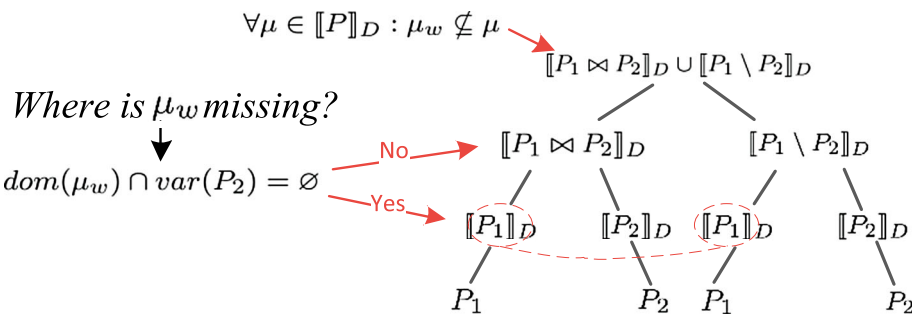


Fig. 7 The parser tree of P

Intuitively, we can use Definition 9 to answer μ_w on P, and $\psi_P = \{\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}\}$. However, note that there are only *UNION* operators between BGPs in Definition 9. $[\![P]\!]_D = [\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$ is a graph pattern with different operators. Thus, we cannot directly apply Definition 9 to $[\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$. In the following, we use reduction to absurdity to proof that $\psi_P \neq \{\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}\}$, where $\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}$ are explanations for μ_w on $P_1 \bowtie P_2$, $P_1 \setminus P_2$, respectively.

Assuming that $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$, $[\![P]\!]_D = [\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$, and $\psi_P = \{\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}\}$.

Then, we compute $\psi_{P_1 \setminus P_2}$ for μ_w on $P_1 \setminus P_2$.

If there exists a mapping $\mu_1 \in [\![P_1]\!]_D$ satisfies $\mu_w \subseteq \mu_1$, we can infer that *MINUS* eliminates μ_1 from $[\![P]\!]_D$ due to $\exists\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$ (cf. analysis in Proposition 1).

With $\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$, we can obtain $(\mu_1 \cup \mu_2) \in [\![P_1 \bowtie P_2]\!]_D$ (cf. Definition 5). Further, with $\mu_w \subseteq \mu_1 \subset (\mu_1 \cup \mu_2)$ and $[\![P]\!]_D = [\![P_1 \bowtie P_2]\!]_D \cup [\![P_1 \setminus P_2]\!]_D$, we can obtain $(\mu_1 \cup \mu_2) \in [\![P]\!]_D : \mu_w \subseteq (\mu_1 \cup \mu_2)$.

$(\mu_1 \cup \mu_2) \in [\![P]\!]_D : \mu_w \subseteq (\mu_1 \cup \mu_2)$ is conflicting with the initial assumption $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$.

If no such a mapping $\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$, μ_1 will belong to $[\![P_1 \setminus P_2]\!]_D$. μ_w will appear in $[\![P]\!]_D$ and conflict with the initial assumption $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$ again.

Hence, $\psi_P \neq \{\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}\}$, where $\psi_{P_1 \bowtie P_2}, \psi_{P_1 \setminus P_2}$ are explanations for μ_w on $P_1 \bowtie P_2$, $P_1 \setminus P_2$, respectively.

(ii) Figure 7 shows the parser tree and query evaluation of $[\![P]\!]_D$. With the premise $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$ (cf. Definition 7) on the root node, we can obtain $\forall\mu \in [\![P_1 \bowtie P_2]\!]_D : \mu_w \not\subseteq \mu$ for the left child of the root, and $\forall\mu \in [\![P_1 \setminus P_2]\!]_D : \mu_w \not\subseteq \mu$ for the right child of the root. For the right child of the root, as described above, there is no possibility that there exists a mapping $\mu_1 \in [\![P_1]\!]_D$ satisfying $\mu_w \subseteq \mu_1$, and $\exists\mu_2 \in [\![P_2]\!]_D : \mu_1 \sim \mu_2$. Therefore, the why-not question on right side of the root can be simplified to $\forall\mu \in [\![P_1]\!]_D : \mu_w \not\subseteq \mu$.

In summary, $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$ can be translated into $\forall\mu \in [\![P_1 \bowtie P_2]\!]_D : \mu_w \not\subseteq \mu$ or $\forall\mu \in [\![P_1]\!]_D : \mu_w \not\subseteq \mu$, depending on whether μ_w contains the variables in P_2, as shown in Fig. 7.

(iii) If $dom(\mu_w) \cap var(P_2) = \varnothing$, then $\forall\mu \in [\![P]\!]_D : \mu_w \not\subseteq \mu$ can be translated into $\forall\mu \in [\![P_1]\!]_D : \mu_w \not\subseteq \mu$, as shown in Fig. 7. Then $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.

Taking Example 6 to illustrate above step, the why-not question $\mu_w = \{?actorname \rightarrow$ "*Spike Lee*"$\}$ and $dom(\mu_w) \cap var(P_2) = \varnothing$. Therefore, $\psi_P = \psi_{P_1}$, and a possible explanation is

$$\psi_{P_1} = \{\langle ?actor\ name\ ?actorname\rangle.$$
$$\langle ?actor\ type\ director\rangle\}.$$

(iv) If $dom(\mu_w) \cap var(P_2) \neq \varnothing$, then $\forall \mu \in [\![P]\!]_D : \mu_w \nsubseteq \mu$ can be translated into $\forall \mu \in [\![P_1 \bowtie P_2]\!]_D : \mu_w \nsubseteq \mu$, as shown in Fig. 7. Then $\psi_P = \psi_{P_1 \bowtie P_2}$, where $\psi_{P_1 \bowtie P_2}$ is the explanation for μ_w on $P_1 \bowtie P_2$.

Taking Example 6 to illustrate above step, the why-not question $\mu'_w = \{?actorname \rightarrow$ "*Leonardo DiCaprio*,"$?country \rightarrow$ "*United States*"$\}$ and $dom(\mu'_w) \cap var(P_2) = \{?country\}$. Thus, $\psi_P = \psi_{P_1 \bowtie P_2}$, and a possible explanation is

$$\psi_{P_1 \bowtie P_2} = \{\langle ?actor\ name\ ?actorname\rangle.$$
$$\langle ?actor\ type\ actor\rangle.$$
$$\langle ?actor\ country\ ?country\rangle.$$
$$\langle ?actor\ gender\ male\rangle.$$
$$\langle ?actor\ haswon\ "Film\ awards\ for\ lead\ actor"\rangle\}.$$

□

Proposition 3 *Let $P = \sigma_R(P_1)$, where P_1 is a BGP, R is a filter condition, the explanation for the why-not question μ_w on P is denoted by ψ_P.*

(1) If $\exists \mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$, then $(FILTER(R), \mu_1) \in \psi_P$, which indicates that μ_1 is eliminated from result mappings by the operator $FILTER$ with the filter condition R.
(2) If $\forall \mu_1 \in [\![P_1]\!]_D : \mu_w \nsubseteq \mu_1$, then $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.

Example 7 Consider a graph pattern $P = \sigma_R(P_1)$ is used to find *all war films since 2000*, where

$$P_1 = \{\langle ?film\ name\ ?filmname\rangle.$$
$$\langle ?film\ type\ War\ films\rangle.$$
$$\langle ?film\ released\ ?date\rangle\},$$

and the filter condition R is $?date >$ "2000."

After graph pattern matching, we may obtain following mappings for P.
$[\![P]\!]_D = \{\{?film \rightarrow f2356, ?filmname \rightarrow$ "*The Pianist*,"
$?date \rightarrow$ "*2000*"$\}$,
$\{?film \rightarrow f3142, ?filmname \rightarrow$ "*Hotel Rwanda*,"
$?date \rightarrow$ "*2004*"$\}$,
$\{?film \rightarrow f1137, ?filmname \rightarrow$ "*Fury*,"
$?date \rightarrow$ "*2014*"$\}\}$.

A why-not question $\mu_w = \{?filmname \rightarrow$ "*Saving Private Ryan*"$\}$ is posed on P.

According to Proposition 3, we should first verify whether $[\![P_1]\!]_D$ contains *Saving Private Ryan* by verifying whether $\exists \mu_1 \in [\![P_1]\!]_D : \{?filmname \rightarrow$ "*Saving Private Ryan*"$\} \subseteq \mu_1$. If $[\![P_1]\!]_D$ contains *Saving Private Ryan*, explaining μ_w will focus on *FILTER*; otherwise, explaining μ_w will focus on P_1.

The following proof for Proposition 3 is illustrated with Example 7.

Proof Let $P = \sigma_R(P_1)$, where P_1 is a BGP, R is a filter condition.

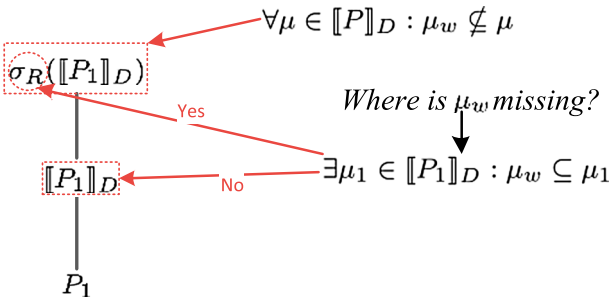


Fig. 8 The parser tree of $P = \sigma_R(P_1)$

(i) According to $[\![\sigma_R(P)]\!]_D = \sigma_R([\![P]\!]_D)$ and $\sigma_R(\Omega) = \{\mu \in \Omega \mid \mu \models R\}$ (cf. Definitions 5, 6), $[\![P]\!]_D = \{\mu_1 \in [\![P_1]\!]_D \mid \mu_1 \models R\}$.

(ii) Figure 8 shows the parser tree and query evaluation of $[\![P]\!]_D$. From bottom to top, the graph pattern matching result $[\![P_1]\!]_D$ will be generated first during the query evaluation, and then, $[\![P_1]\!]_D$ will be constrained by σ_R to generate the final $[\![P]\!]_D$. With the premise $\forall \mu \in [\![P]\!]_D : \mu_w \nsubseteq \mu$ (cf. Definition 7), we can observe that μ_w can be eliminated in two cases: eliminated by the *FILTER* operator, or eliminated by the graph pattern matching of P_1. We can locate where μ_w is missing on the parser tree by determining whether $\exists \mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$.

(iii) If $\exists \mu_1 \in [\![P_1]\!]_D : \mu_w \subseteq \mu_1$, taking a look at the parser tree in Fig. 8 μ_w is contained by $\mu_1 \in [\![P_1]\!]_D$. However, on the root of the parser tree, μ_1 does not satisfy R and *FILTER* eliminates μ_1 from result mappings, hence, $(FILTER(R), \mu_1) \in \psi_P$.

Taking Example 7 to illustrate above step, we may obtain four mappings for P_1.

$[\![P_1]\!]_D = \{\{?film \rightarrow f2356, ?filmname \rightarrow$ "*The Pianist*,"
$?date \rightarrow$ " *2000*"$\}$,
$\{?film \rightarrow f3142, ?filmname \rightarrow$ "*Hotel Rwanda*,"
$?date \rightarrow$ " *2004*"$\}$,
$\{?film \rightarrow f1137, ?filmname \rightarrow$ "*Fury*,"
$?date \rightarrow$ "*2014*"$\}$,
$\{?film \rightarrow f2209, ?filmname \rightarrow$ "*Saving Private Ryan*,"
$?date \rightarrow$ "*1998*"$\}\}$.

We can find that $\mu_w = \{?filmname \rightarrow$ "*Saving Private Ryan*"$\}$ is contained by $\{?film \rightarrow f2209, ?filmname \rightarrow$ "*Saving Private Ryan*,"$?date \rightarrow$ "*1998*"$\}$ in $[\![P_1]\!]_D$, and the reason for missing μ_w is that *FILTER* eliminates the mapping $\{?film \rightarrow f2209, ?filmname \rightarrow$ "*Saving Private Ryan*," $?date \rightarrow$ "*1998*"$\}$ from the result set because of "*1998*" $<$ "*2000*." Therefore,

$$
\begin{aligned}
&(\text{FILTER}(?date > \text{“2000”}),\\
&\{?film \rightarrow f2209,\\
&?filmname \rightarrow \text{“}Saving\ Private\ Ryan,\text{”}\\
&?date \rightarrow \text{“1998”}\})
\end{aligned}
$$

will be added to ψ_P.

(iv) If $\forall \mu_1 \in [\![P_1]\!]_D : \mu_w \nsubseteq \mu_1$, the reason for μ_w is that the graph pattern matching of P_1 eliminates μ_1 from result mappings. Taking a look at the parser tree in Fig. 8, $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1.

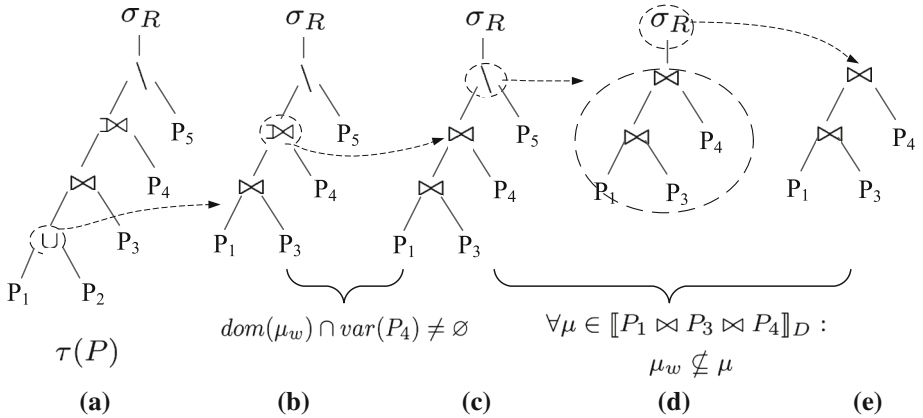


Fig. 9 Explaining μ_w on the parse tree $\tau(P)$

Taking Example 7 to illustrate above step, we may obtain three mappings for P_1.

$[\![P_1]\!]_D = \{\{?film \rightarrow f2356, ?filmname \rightarrow$ "*The Pianist*,"
$?date \rightarrow$ " *2000*"},
$\{?film \rightarrow f3142, ?filmname \rightarrow$ "*Hotel Rwanda*,"
$?date \rightarrow$ " *2004*"},
$\{?film \rightarrow f1137, ?filmname \rightarrow$ "*Fury*,"
$?date \rightarrow$ "*2014*"}}.

Then we can observe that none of $[\![P_1]\!]_D$ contains $\mu_w = \{?filmname \rightarrow$ "*Saving Private Ryan*"}. Hence, μ_w on $P = \sigma_R(P_1)$ can be simplified to μ_w on P and $\psi_P = \psi_{P_1}$, where ψ_{P_1} is the explanation for μ_w on P_1. A possible explanation is

$$\begin{aligned}\psi_P = \psi_{P_1} = \{&\langle ?film\ name\ ?filmname\rangle.\\ &\langle ?film\ released\ ?date\rangle.\\ &\langle ?film\ type\ film\rangle\}.\end{aligned}$$

□

In summary, for why-not questions on *MINUS* and *FILTER*, we should consider both operators and inappropriate triple patterns in BGPs according to *Proposition* 1 and *3*; for why-not questions on *OPTIONAL*, we should focus on inappropriate triple patterns in BGPs according to *Proposition* 2.

Next, we investigate the complex scenario when graph patterns blend together with different operators. Given BGPs P_1, P_2, P_3, P_4 and P_5, a graph pattern $P = \sigma_R(\ (\ (\ (P_1 \cup P_2) \bowtie P_3) \ltimes P_4\ ) \setminus P_5\ )$, a why-not question μ_w on P and $dom(\mu_w) \cap var(P_4) \neq \varnothing$. Figure 9a shows the parse tree $\tau(P)$.

First, the operator *UNION* in $\tau(P)$ means that $\tau(P)$ needs to be converted to different cases for μ_w by using the distributivity equivalences

$$\begin{aligned}(P_1 \cup P_2) \bowtie P_3 &= (P_1 \bowtie P_3) \cup (P_2 \bowtie P_3),\\ P_1 \bowtie (P_2 \cup P_3) &= (P_1 \bowtie P_2) \cup (P_1 \bowtie P_3),\\ (P_1 \cup P_2) \ltimes P_3 &= (P_1 \ltimes P_3) \cup (P_2 \ltimes P_3),\\ (P_1 \cup P_2) \setminus P_3 &= (P_1 \setminus P_3) \cup (P_2 \setminus P_3).\end{aligned} \tag{3}$$

Figure 9b presents one of the cases with P_1. Then, $\rtimes$ will be translated into $\bowtie$ because of $dom(\mu_w) \cap var(P_4) \neq \varnothing$ according to *Proposition* 2, as shown in Fig. 9b, c. Next, if the evaluation $[\![P_1 \rtimes P_3 \rtimes P_4]\!]$ of *MINUS*'s left side does not contain μ_w (i.e., $\forall \mu \in [\![P_1 \rtimes P_3 \rtimes P_4]\!]_D : \mu_w \nsubseteq \mu$), the operator *MINUS* and its right side P_5 will be removed from $\tau(P)$ according to *Proposition* 1, as shown in Fig. 9c, d. Finally, for the operator *FILTER*, μ_w is simplified to μ_w on $P_1 \rtimes P_3 \rtimes P_4$ because of $\forall \mu \in [\![P_1 \rtimes P_3 \rtimes P_4]\!]_D : \mu_w \nsubseteq \mu$ according to *Proposition* 3, as shown in Fig. 9d, e.

In the above Example, $\exists \mu \in [\![P_1 \rtimes P_3 \rtimes P_4]\!]_D : \mu_w \subseteq \mu$ is a necessity for μ_w appearing in $[\![P]\!]_D$. We further define a new important notion, the necessary BGP, which is used frequently in the following sections.

Definition 10 (*Necessary BGP*) Consider the SPARQL query Q (*SELECT S WHERE P*), a why-not question μ_w on Q, the necessary BGP for μ_w is a basic graph pattern NB, where $NB \subseteq P$ and $\exists \mu_N \in [\![NB]\!]_D : \mu_w \subseteq \mu_N$ is a necessity for μ_w appearing in $[\![Q]\!]_D$.

For instance, Fig. 9e illustrates the necessary BGP for the why-not question μ_w in the above example.

According to the above analyses, ANNA is designed to address why-not questions on SPARQL queries using a divide-and-conquer strategy. ANNA first computes the necessary BGP and identifies where the absence of the expected items occurs and then provides explanations according to different reasons.

5 Explanation model

5.1 Framework

Given an RDF dataset D, a SPARQL query Q and a why-not question μ_w on Q, Fig. 10 illustrates the framework of generating a set of explanations Ψ by ANNA, which mainly includes three modules: **computing the necessary BGP** (described in Sect. 5.2), **modifying graph patterns** (described in Sect. 5.3) and **identifying inappropriate operators** (described in Sect. 5.4).

Module I (computing the necessary BGP) This module first generates a parse tree $\tau(P)$ of the SPARQL query Q and then computes the necessary BGP NB for the why-not question μ_w based on $\tau(P)$ (described in Sect. 5.2). If $\tau(P)$ involves *UNION* operators, then this module will output a set of necessary BGPs NBs, and ANNA will explain μ_w on each necessary BGP in NBs, respectively.

With a necessary BGP NB, ANNA identifies where the expected mapping μ_w is removed by determining $\exists \mu_{NB} \in [\![NB]\!]_D : \mu_w \subseteq \mu_{NB}$, as shown in Fig. 10. If such a μ_{NB} exists, then the why-not reason will be located at inappropriate operators; otherwise, it will be located at inappropriate triple patterns in NB.

To determine $\exists \mu_{NB} \in [\![NB]\!]_D : \mu_w \subseteq \mu_{NB}$, a new BGP NB' will be generated from NB by substituting variables according to μ_w. For example, assuming a necessary BGP

$$NB_1 = \{\langle ?film\ name\ \ ?filmname\rangle.\ \langle ?film\ director\ ?person\rangle.\ \langle ?film\ released\ ?date\rangle\},$$

and a why-not question $\{?filmname \rightarrow \text{“}Batman''\}$,

$$NB_1' = \{\langle ?film\ name\ \text{“}Batman''\rangle.$$

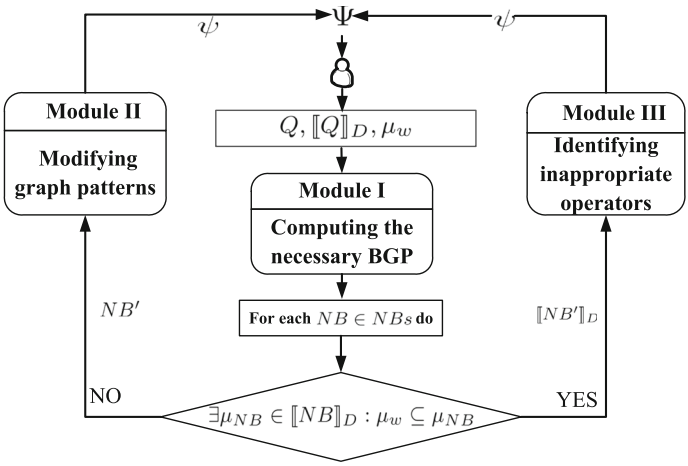


Fig. 10 Framework of ANNA

$\langle ?film\ director\ ?person \rangle$.
$\langle ?film\ released\ ?date \rangle\}$.

$[\![NB']\!]_D$ can be returned by issuing NB' to the SPARQL query engine and $[\![NB']\!]_D \subseteq [\![NB]\!]_D$. If $[\![NB']\!]_D \neq \varnothing$, then $\forall \mu_{NB'} \in [\![NB']\!]_D : \mu_w \subseteq \mu_{NB'}$, which means that $\exists \mu_{NB} \in [\![NB]\!]_D : \mu_w \subseteq \mu_{NB}$; otherwise, it suggests that $\forall \mu_{NB} \in [\![NB]\!]_D : \mu_w \nsubseteq \mu_{NB}$. NB' will be the input of the next modules instead of NB. If μ_w is about why the result is an empty set, i.e., $[\![NB]\!]_D = \varnothing$, then $NB' = NB$.

Module II (modifying graph patterns) For the why-not question μ_w caused by inappropriate triple patterns in NB', this module generates a modified graph pattern mNB' by a graph-based approach (described in Sect. 5.3). mNB' should be similar to NB' and can be utilized to match RDF graphs for the expected mapping μ_w. Then, mNB' will be returned as an explanation ψ.

Module III (identifying inappropriate operators) For a why-not question μ_w caused by inappropriate operators, this module will compare each mapping $\mu_{NB'} \in [\![NB']\!]_D$ with the output of questionable operator via post-order traversal on the parse tree $\tau(P)$ of Q. Once a mapping $\mu_{NB'}$ ($\mu_w \subseteq \mu_{NB'}$) is filtered out by an operator op, the module will add a tuple $(op, \mu_{NB'})$ to the explanation ψ.

Algorithm 1 highlights the main steps of generating explanations. Lines 4–11 are to determine $\exists \mu_{NB} \in [\![NB]\!]_D : \mu_w \subseteq \mu_{NB}$. Lines 5–10 are to replace variables of tp according to μ_w and generate NB'. Function *ComputingNB* (Module I) is described in Sect. 5.2, *ModifyingGP* (Module II) is described in Sect. 5.3, and *InappropriateOperator* (Module III) is described in Sect. 5.4.

5.2 Computing the necessary BGP

The module of computing the necessary BGP (Module I) aims to find the necessary BGP NB for μ_w. In this module, the parse tree $\tau(P)$ is generated from Q, and computing the necessary BGP essentially involves eliminating $\cup, \setminus, \bowtie, \sigma$ from $\tau(P)$ and retaining the basic graph patterns in $\tau(P)$, which must match RDF triples for μ_w.

Algorithm 1 Generating an explanation

Input: SPARQL query Q, $[\![Q]\!]_D$, μ_w
Output: A set of explanations Ψ
1: Initialization: $NBs = \varnothing$, $\Psi = \varnothing$;
2: $NBs \leftarrow$ComputingNB(Q);
3: **for all** $NB \in NBs$ **do**
4: $NB' = \varnothing$;
5: **for all** $tp \in NB$ **do**
6: **if** tp.contains($?w \in \mu_w$) **then**
7: $tp.?w = w$
8: **end if**
9: NB'.add(tp);
10: **end for**
11: **if** $[\![NB']\!]_D \neq \varnothing$ **then**
12: $\psi \leftarrow$InappropriateOperator (Q, $[\![NB']\!]_D$) ;
13: **else**
14: $\psi \leftarrow$ ModifyingGP(NB', μ_w) ;
15: **end if**
16: Ψ.add(ψ);
17: **end for**
18: **return** Ψ.

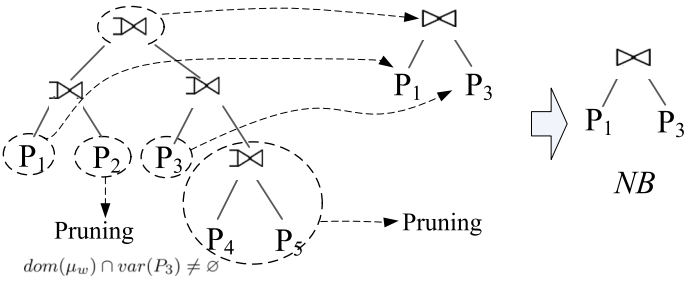


Fig. 11 Eliminating *OPTIONALs* from the parse tree $\tau(P)$

As analyzed in Sect. 4.2, if $\tau(P)$ involves *UNION*s, then NB needs to be processed according to Equation (3). However, the following distributivity equivalences do not hold when $\cup$ is in the right side of $\setminus$ or $\bowtie$ [28],

$$\begin{aligned} P_1 \bowtie (P_2 \cup P_3) &= (P_1 \bowtie P_2) \cup (P_1 \bowtie P_3), \\ P_1 \setminus (P_2 \cup P_3) &= (P_1 \setminus P_2) \cup (P_1 \setminus P_3). \end{aligned} \tag{4}$$

Hence, *OPTIONAL* and *MINUS* operators should be eliminated from the parse tree $\tau(P)$ by pre-order traversal before the *UNION* is to be processed.

According to *Proposition* 2, once a node is $\bowtie$, if the right-side graph pattern of $\bowtie$ contains variables of μ_w, then the $\bowtie$ will be translated into $\bowtie$; otherwise, the right-side graph pattern of $\bowtie$ will be pruned, and $\bowtie$ itself will be replaced by its left-side graph pattern.

For instance, a graph pattern P is $(P_1 \bowtie P_2) \bowtie (P_3 \bowtie (P_4 \bowtie P_5))$ and $dom(\mu_w) \cap var(P_3) \neq \varnothing$. Figure 11 illustrates the eliminations of *OPTIONAL* operators from the parse tree $\tau(P)$. The necessary BGP NB is $P_1 \bowtie P_3$.

According to *Proposition* 1, once a node is $\setminus$, the right-side graph pattern of $\setminus$ will be pruned, and $\setminus$ itself will be replaced by its left-side graph pattern.

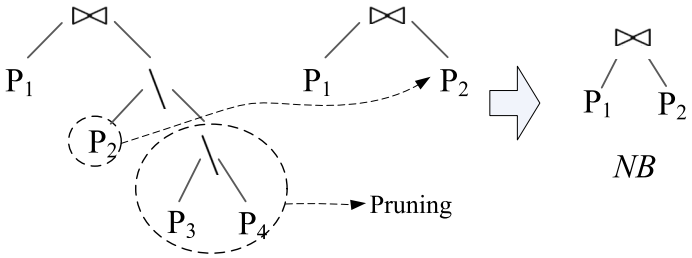


Fig. 12 Eliminating *MINUSs* from the parse tree $\tau(P)$

Table 1 Statistics of four SPARQL query log datasets in LSQ

Dataset	Date	#Queries
DBpedia	30/04/2010 to 20/07/2010	1.7 million
Linked Geo Data (LGD)	24/11/2010 to 06/07/2011	1.6 million
Semantic Web Dog Food (SWDF)	16/05/2014 to 12/11/2014	1.4 million
British Museum (BM)	08/11/2014 to 01/12/2014	800 thousands

For instance, a graph pattern P is $P_1 \bowtie (P_2 \setminus (P_3 \setminus P_4))$. Figure 12 illustrates the eliminations of *MINUS* operators from the parse tree $\tau(P)$. The necessary BGP NB is $P_1 \bowtie P_2$.

Given a SPARQL query Q and the why-not question μ_w, we propose the algorithm *Computing necessary BGP* to compute NB for μ_w, whose pseudocode is presented in Algorithm 2. First, the parse tree $\tau(P)$ is constructed by parsing Q [28] (Line 2). Next, *FILTER*, *OPTIONAL* and *MINUS* operators are eliminated by pre-order traversal on $\tau(P)$ (Lines 3–29). Then, according to Equation (3), the *UNION* operators are processed to convert $\tau(P)$ from conjunctive normal form (CNF) to disjunctive normal form (DNF) (Line 30). Finally, the remaining graph patterns are divided into basic graph patterns by *UNIONs*, and each BGP forms a necessary BGP (Lines 31–32).

The algorithm 2 of Module I (computing the necessary BGP) mainly consists of two parts: 1) eliminating *FILTER, OPTIONAL* and *MINUS* operators by pre-order traversal on parser tree $\tau(P)$ [28]; 2) processing *UNION* operators to convert $\tau(P)$ from conjunctive normal form (CNF) to disjunctive normal form (DNF). For the pre-order traversal, the time complexity is $O(n)$, where n is the number of operators in $\tau(P)$. For processing *UNION* operators, the process of converting CNF to DNF has been proved NP-hard [27]. And its worst time complexity is $O(2^m)$; m is the number of *UNION*s. Hence, the time complexity of module is $O(n + 2^m)$.

We analyze SPARQL queries from the logs of four datasets in LSQ [29] shown in Table 1. We find that the *UNION* queries are not often used in DBpedia, LGD and BM. The percentage of *UNION* queries is 4.42% in DBpeida, 9.65% in LGD, 0.00% in BM, respectively. Among these *UNION* queries, 96.25% in DBpedia (99.11% in LGD) have one *UNION*. Although the percentage of *UNION* queries in SWDF is 32.71%, but 98.77% of them contain one single *UNION* only, 0.93% have two *UNION*s. Hence, the number of *UNION*s is usually very tiny (and frequently is just one) in the real-world SPARQL queries, and we can ignore this cost in most time.

Algorithm 2 Computing necessary BGP

Input: SPARQL query Q
Output: A set of necessary BGP NBs

Initialization: $NBs = \varnothing$, $\tau(P) = \varnothing$, $\psi = \varnothing$, $Stack$;
$\tau(P) \leftarrow$ ParseQuery (Q); // return a parse tree
$Node \leftarrow$SelectRootNode($\tau(P)$);
while $Node$!= NULL or !$Stack$.Empty() **do**
  while $Node$!=NULL **do**
    if $Node$ is σ **then**
      //eliminating σ according to *Proposition* 3
      $Node$=$Node \rightarrow leftchild$;
    end if
    if $Node$ is $\setminus$ **then**
      //eliminating $\setminus$ according to *Proposition* 1
      $Node \rightarrow rightchild$=$\varnothing$;
      $Node$=$Node \rightarrow leftchild$;
    end if
    if $Node$ is $\rtimes$ **then**
      //eliminating $\rtimes$ according to *Proposition* 2
      if $dom(\mu_w) \cap var(Node.RigthTree) = \varnothing$ **then**
        $Node \rightarrow rightchild$=$\varnothing$;
      end if
      $Node$=$Node \rightarrow leftchild$;
    end if
    $Stack$.push($Node$);
    $Node$=$Node \rightarrow leftchild$;
  end while
  if !$Stack$.Empty() **then**
    $Node \leftarrow Stack$.pop();
    $Node$=$Node \rightarrow rightchild$;
  end if
end while
$\tau(P) \leftarrow$applying Equation (3) on $\tau(P)$;
//extracting BGPs from $\tau(P)$ divided by $\cup$.
$NBs \leftarrow$ExtractingBGPs ($\tau(P)$);
return $\tau(P)$.

We also measure the execution time of Module I answering 61 why-not questions in our experiment, and the average time is 1.3 s. This amount of time is acceptable for the whole framework and indicates that *UNION*s have minimal impact on the efficiency of module I.

5.3 Modifying graph patterns

The module of modifying graph patterns (Module II) aims to address why-not questions caused by inappropriate triple patterns in Q. The input of this module is NB' ($[\![NB']\!]_D = \varnothing$). To explain μ_w, this module will detect and modify inappropriate triple patterns in NB'. The modified BGP mNB' of NB' will be returned as an explanation and help refine the initial BGPs of SPARQL queries, which are illustrated with Example 8 as follows:

Example 8 Consider a necessary BGP NB_4 is used to find the actors who starred in *Tim Burton*'s film *Alice in Wonderland* and starred in another *Tim Burton* film.

$$NB_4 = \{\langle ?actor\ name\ ?actorname\rangle.$$
$$\langle ?actor\ strarring\ ?film1\rangle.$$

⟨?*actor strarring* ?*film2*⟩.
⟨?*film1 name* "*Alice in Wonderland*"⟩.
⟨?*person director* ?*film1*⟩.
⟨?*person name* "Tim Burton"⟩.
⟨?*person director* ?*film2*⟩.
⟨?*film2 name* ?*film2name*⟩}.

The user proposes a why-not question μ_w =
{?*actorname* → "*Alan Rickman*,"
?*film2name* → "*Alice Through the Looking Glass*"}.

NB' has a graph structure, in which triple patterns are connected by $\bowtie$. Figure 13a shows a graphical representation of NB'_4 generated from NB_4 according to μ_w ($[\![NB'_4]\!]_D = \varnothing$), and $tp_1 \sim tp_8$ are the identifiers of the triple pattern in NB'_4. A naive solution to find inappropriate triple patterns is examining the output of joined triple patterns in a left linear fashion. This solution may lead to an exhaustive search because the time cost of each identification step is proportional to the number of RDF triples that are produced during examination, which has been proved in [30]. For instance, tp_1 and tp_2 are used to find all actors and their starring films in the dataset, and they will produce 81,693,233 RDF triples which need to be joined with each other and examined one by one.

To tackle the above challenge, ANNA employs a novel graph-based approach to detect inappropriate triple patterns. The approach considers the topological properties and mainly includes three steps: BGP partition, inner-part modification and inter-part modification. We will illustrate these three steps with Example 8 in next sections.

5.3.1 BGP partition

This step aims to partition the graph pattern of Q into several parts based on different types of $\bowtie$.

For a BGP, $\bowtie$ between two triple patterns is on the shared subjects or objects. Thus, the typical types of $\bowtie$ are subject–subject $\bowtie$ (S-S $\bowtie$), subject–object $\bowtie$ (S-O $\bowtie$) and object–object $\bowtie$ (O-O $\bowtie$). In this study, we do not consider $\bowtie$ on predicate, because it is uncommon [32]. Triple patterns joined by S-S $\bowtie$ typically describe an entity and the property values to be queried, e.g., ⟨*?film name ?filmname*⟩ $\bowtie$ ⟨*?film director ?person*⟩. Triple patterns joined by S-O $\bowtie$ or O-O $\bowtie$ describe the relationships between two different entities to be queried, e.g., ⟨*?film director ?person*⟩ $\bowtie$ ⟨*?person name*"Tim Burton"⟩. The empirical study [32] on real-world SPARQL queries from DBpedia logs also reveals that the most common type of $\bowtie$ is S-S $\bowtie$ (59.23%), followed by S-O $\bowtie$ (35.88 %) and O-O $\bowtie$ (4.66 %). This finding indicates that entities and their property values are more concerned by users, and triple patterns joined by S-S $\bowtie$ should be detected first.

On the basis of different types of $\bowtie$, NB' can be partitioned into several parts. Each part $Part_i \subseteq NB'$ contains only S-S $\bowtie$ and connects to others by S-O $\bowtie$ or O-O $\bowtie$. Figure 14 illustrates the partition of NB'_4. With the BGP partition, ANNA will first detect and modify the triple patterns shared on the same subject, which are more likely to appear in SPARQL queries. On the other hand, ANNA can detect inappropriate triple patterns in a parallel process, i.e., each part after partitioning can be parallelized onto the threads (or computing nodes) without interactions.

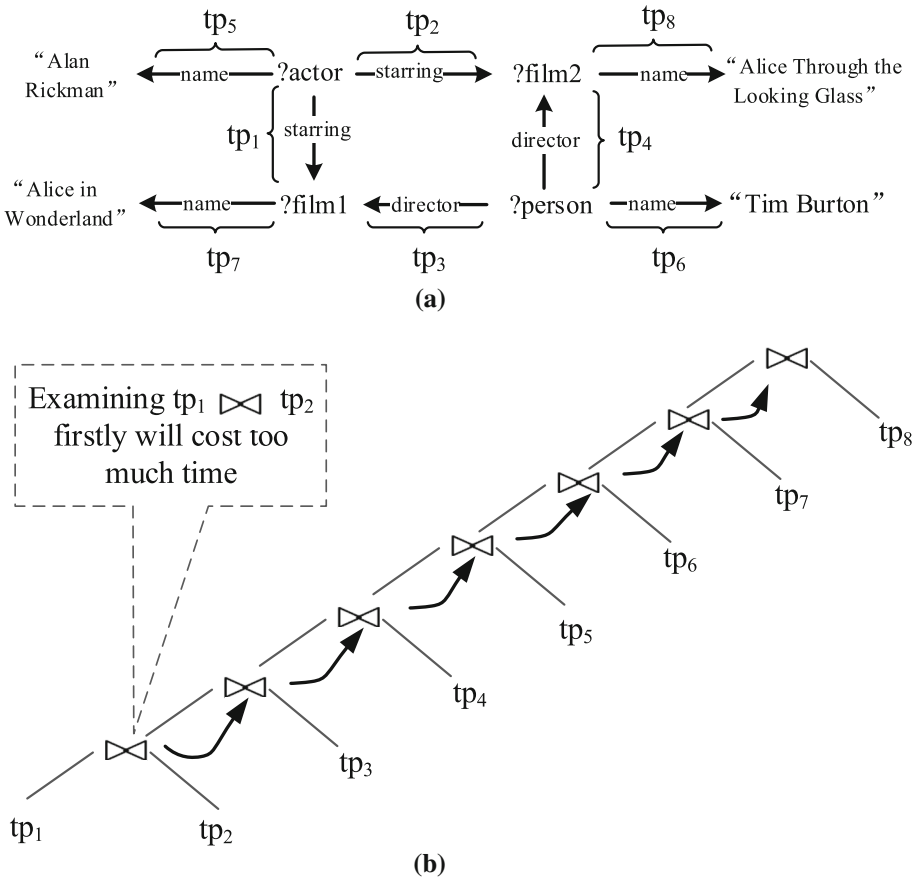


Fig. 13 NB'_4 of necessary BGP NB_4, and identification process of joined triple patterns. **a** NB'_4, **b** Identifying the joined pattern in a linear fashion

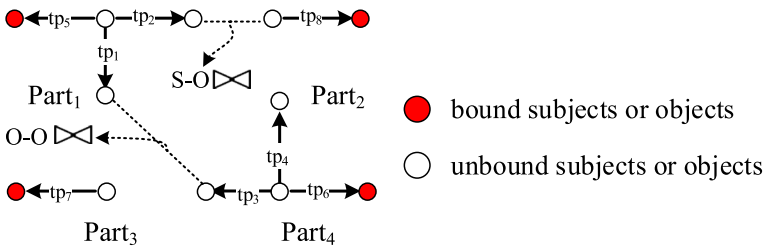


Fig. 14 The partition of NB'_4

5.3.2 Inner-part modification

This step aims to modify inappropriate directions and RDF terms in triple patterns shared on the same subject. For each $Part_i \subseteq NB'$, ANNA detects each triple pattern $tp_j \in Part_i$ by adding tp_j to a detecting BGP $dPart_i$ in an orderly manner and issuing $dPart_i$ to D. If $[\![dPart_i]\!]_D \neq \varnothing$, then tp_j will be marked as normal. If $[\![dPart_i]\!]_D = \varnothing$, then tp_j is an

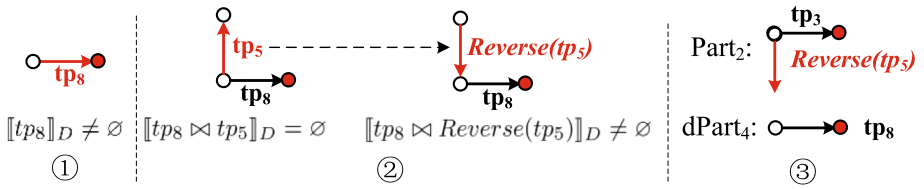


Fig. 15 Detection and modification process of tp_5 in NB'_4

inappropriate triple pattern and should be modified. Note that RDF terms specified by μ_w in tp_j should not be modified. The subject of tp_j shared by other triple patterns in $Part_i$ cannot be modified.

Now, we describe the modifications for the inner-part inappropriate triple patterns. Given an inappropriate triple pattern $tp_j \in Part_i$ and $Part_i \subseteq NB'$, $[\![dPart_i]\!]_D = \varnothing$ after adding tp_j to $dPart_i$.

Direction Modification ANNA uses $Reverse()$ (cf. Sect. 4.1) and verifies whether $[\![dPart_i]\!]_D = \varnothing$ after adding $Reverse(tp_j)$ into $dPart_i$. If $[\![dPart_i]\!]_D \neq \varnothing$, ANNA will further determine whether $\exists Part_z \subseteq NB'$, which connects to $Reverse(tp_j)$ with S-S $\bowtie$. If such $Part_z$ exists, then $Reverse(tp_j)$ will be added to $Part_z$ to make the inner-part modification; otherwise, $Reverse(tp_j)$ will be taken out from $dPart_i$ to make the inter-part modification in the next step.

For instance, tp_5 of $Part_4$ is in an inappropriate direction, as shown in Fig. 14. Figure 15 illustrates the detection and modification process from Steps 1 to 3 with tp_5. In Step 1, $[\![tp_8]\!]_D \neq \varnothing$ indicates that tp_8 is normal. In Step 2, tp_5 is detected as an inappropriate triple pattern and $Reverse(tp_5)$ is an effective modification. In Step 3, $Reverse(tp_5)$ is added to $Part_2$, and $dPart_4$ is ready to perform the next detection.

If $[\![NB]\!]_D \neq \varnothing$, then all directions of predicates in NB are correct (cf. analysis of AND in Sect. 4.2) and $Reverse()$ will not be applied to any triple patterns in $Part_i \subseteq NB'$.

RDF term modification If $Reverse(tp_j)$ is added into $dPart_i$ but $[\![dPart_i]\!]_D$ is still an empty set, then ANNA will use $Modify()$ to modify an RDF term in tp_j and generate a new tp'_j, which satisfies $[\![dPart_i]\!]_D \neq \varnothing$ after adding tp'_j to $dPart_i$. $Modify()$ is implemented by the ontology-based approach, which generates a modified RDF term r' by performing RDFS inference rules and ontology on original RDF term r. For example, *director* in NB'_4 can be modified to *producer* by applying subproperty rule[7] with the *film* ontologies in *DBpedia*.

The semantic similarity between r and r' can be measured by Wu and Palmer similarity [27] and is defined as

$$sim(r, r') = \frac{2depth(LCA(r, r'))}{depth(r) + depth(r')}, \tag{5}$$

where $LCA(r, r')$ is the least common ancestor of r and r' in the ontology of D and $depth(r)$ is the depth of r in the ontology. If r' is a variable, then $sim(r, r') = 0$.

For inappropriate RDF terms of $tp_j = \langle s_j, p_j, o_j \rangle$, three cases exist for the modification of tp_j.

Case 1 p_j and o_j are already bound to an RDF term in D. Thus, both p_j and o_j can be modified. ANNA will then choose the RDF term r^* which satisfies

$$r^* = \arg\max_{r \in \{o_j, p_j\}} sim(r, Modify(r)), \tag{6}$$

[7] http://www.w3.org/TR/rdf-schema/.

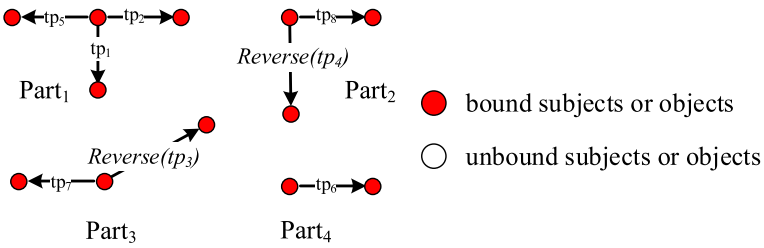


Fig. 16 Inner-part modification of NB'_4

and use $Modify(r^*)$ to generate the new triple pattern tp'_j.

Case 2 p_j is bound, o_j is a variable. ANNA will use $Modify(p_j)$ to generate the new triple pattern tp'_j.

Case 3 p_j is a variable, o_j is bound. ANNA will use $Modify(o_j)$ to generate the new triple pattern tp'_j.

Efficiency optimization To improve the efficiency of inner-part modification, the procedure of detecting triple patterns has three strategies:

(i) Triple patterns that contain expected RDF terms of μ_w should be detected first (e.g., tp_1 in Example 8), because ANNA should first determine triple patterns that contain expected RDF terms which are normal.

(ii) If a triple pattern $tp_j \in Part_i$ is inappropriate, then ANNA will modify tp_j to tp'_j and determine $[\![Part_i]\!]_D \neq \varnothing$ (note that this is not $[\![dPart_i]\!]_D \neq \varnothing$) with the new tp'_j. If $[\![Part_i]\!]_D \neq \varnothing$, ANNA will terminate the detection of $Part_i$.

(iii) The efficiency of the detecting procedure varies by the orders of selecting triple patterns to detect. The order of selecting triple patterns to be detected impacts the efficiency of detection procedure. To improve the efficiency, we implemented the heuristic in [30] for the selection order of triple patterns:

$$\langle s_j, p_j, o_j\rangle \prec \langle ?, p_j, o_j\rangle \prec \langle s_j, p_j, ?\rangle \prec \langle s_j, ?, o_j\rangle$$
$$\prec \langle ?, p_j, ?\rangle \prec \langle ?, ?, o_j\rangle \prec \langle s_j, ?, ?\rangle \prec \langle ?, ?, ?\rangle,$$

where the triple pattern on the left side of $\prec$ is more selective than the one on the right side of $\prec$, s_j, p_j, o_j represent subject, predicate and object, and ? refers to a variable. In the above heuristic, (1) the fewer variables a triple pattern has, the more selective it is likely to be; (2) a triple pattern with unbound subjects is more selective than a triple pattern with unbound objects or predicates.

After all triple patterns are detected, the unbound variable $?x \in Part_i$ will be bounded according to $\pi_{?x}[\![Part_i]\!]_D$. Figure 16 shows the result of inner-part modification of NB'_4. All subjects and objects are bound.

5.3.3 Inter-part modification

After inner-part detection and modification of triple patterns, this step aims to modify inappropriate RDF terms in triple patterns which connect different parts of NB'.

ANNA first merges $Part_i \subseteq NB'$ by processing the S-O $\bowtie$ or O-O $\bowtie$ between different parts. Consider a S-O $\bowtie$ or O-O $\bowtie$ between $Part_i$ and $Part_z$ of NB', their shared subject or object is denoted by c. After inner-part modifications, c is already bounded by $\pi_c[\![Part_i]\!]_D$

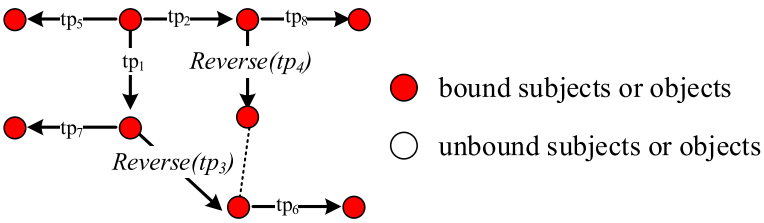


Fig. 17 Merging sub parts of NB'_4

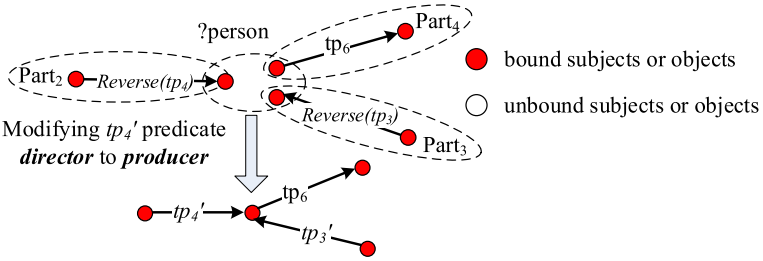


Fig. 18 Inter-part modification of $NB'4$

in $Part_i$ and $\pi_c[\![Part_z]\!]_D$ in $Part_z$. If $\pi_c[\![Part_z]\!]_D \bowtie \pi_c[\![Part_i]\!]_D \neq \varnothing$, ANNA will merge $Part_i$ and $Part_z$ by connecting the related triple patterns with S-O $\bowtie$ or O-O $\bowtie$, and c will be bounded by $\pi_c[\![Part_z]\!]_D \bowtie \pi_c[\![Part_i]\!]_D$ before detecting the next inter-part connection. Figure 17 shows the result of merging all $Part_i \subseteq NB'_4$ in Example 8.

For the unconnected S-O $\bowtie$ and O-O $\bowtie$, ANNA will locate the shared node c in NB'. For each predicate p that connects to c, ANNA will use $Modify()$ to modify p to p', which satisfies all S-O $\bowtie$, and O-O $\bowtie$ on c can be merged together if p is replaced with p'. ANNA will choose the predicate p^* satisfying

$$p^* = \underset{p \in \{tp | c \in tp \wedge tp \in Part_i\}}{\arg\max} sim(p, Modify(p)), \tag{7}$$

and use $Modify(p^*)$ to modify NB'.

For instance, *?person* is shared by $Part_2$, $Part_3$ and $Part_4$ in NB'_4, as shown in Fig. 18. *?person* fails to connect $Part_2$, $Part_3$ and $Part_4$ because $\pi_{?person}[\![Part_2]\!]_D \bowtie \pi_{?person}[\![Part_3]\!]_D \bowtie \pi_{?person}[\![Part_4]\!]_D = \varnothing$. $Part_2$, $Part_3$, and $Part_4$ will be connected by *?person* if the *director* of tp_5 is modified to *producer*, and $\pi_{?person}[\![Part_2]\!]_D \bowtie \pi_{?person}[\![Part_3]\!]_D \bowtie \pi_{?person}[\![Part_4]\!]_D \neq \varnothing$.

Finally, a modified BGP mNB' of NB' will be returned as the explanation for the why-not question to the user, with which the user can refine the initial BGP NB. Figure 19 shows the mNB'_4 for NB'_4, which indicates that the *director* of *Alice Through the Looking Glass* is not *Tim Burton*, and the direction of *director* in initial SPARQL query is inappropriate.

5.4 Identifying inappropriate operators

The module of identifying inappropriate operators (Module III) aims to address why-not questions caused by inappropriate operators. The input of this module is a set of mappings $[\![NB']\!]_D$. Each mapping $\mu_{NB'} \in [\![NB']\!]_D$ contains the expected mapping μ_w.

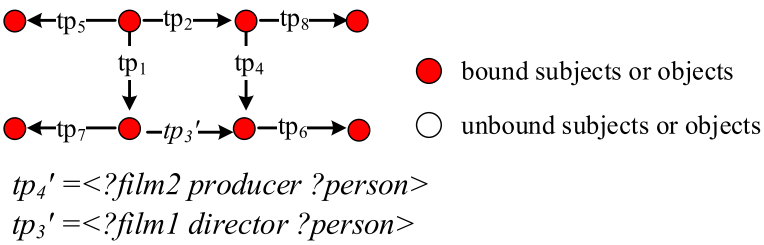


Fig. 19 The modified BGP $mNB'4$

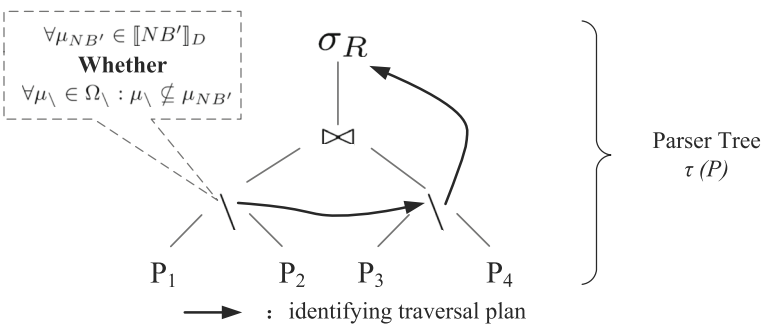


Fig. 20 Parser tree $\tau(P)$ of Q_5 in Example 9

As analyzed in Sect. 4.2, *MINUS* and *FILTER* are only two questionable operators that can eliminate expected mappings from the query processing. Therefore, ANNA will only compare each mapping $\mu_{NB'} \in [\![NB']\!]_D$ with *MINUS* or *FILTER*'s output via post-order traversal on the parse tree $\tau(P)$ of Q.

Given a SPARQL query Q (*SELECT S WHERE P*), a why-not question μ_w, the necessary BGP NB for μ_w, and $[\![NB']\!]_D \neq \varnothing$, we propose the algorithm *Identifying Inappropriate Operators*, whose pseudocode is presented in Algorithm 3. The identification procedure is illustrated with Example 9 as follows:

Example 9 Given a SPARQL query Q_5, the graph pattern P of Q_5 is $\sigma_R((P_1 \setminus P_2) \bowtie (P_3 \setminus P_4))$. The necessary BGP NB is $P_1 \bowtie P_3$ and NB' is generated from NB according to μ_w, $[\![NB']\!]_D \neq \varnothing, \forall \mu_{NB'} \in [\![NB']\!]_D : \mu_w \subseteq \mu_{NB'}$.

First, the parse tree $\tau(P)$ is constructed by parsing Q [28] (Line 2). For instance, the $\tau(P)$ of Q_5 in Example 9 is shown in Fig. 20.

Next, a multiset [15] of *FILTER* and *MINUS* operators, OP, can be screened from $\tau(P)$ by post-order traversal on $\tau(P)$ (Line 3). As shown in Fig. 20, OP of Q_5 in Example 9 is { *MINUS* \, *MINUS* \, *FILTER* σ}.

Lines 4–18 identify inappropriate operators. For each $op \in OP$, its output Ω_{op} is a set of mappings (Line 5). For $\mu_{NB'} \in [\![NB']\!]_D$, if $\forall \mu_{op} \in \Omega_{op} : \mu_{op} \not\subseteq \mu_{NB'}$, then op filters out $\mu_{NB'}$ from query processing (Line 7).

If the inappropriate op is *FILTER* with the filter condition R, then the tuple $(FILTER(R), \mu_{NB'})$ will be added into the explanation ψ (Lines 8–10).

If op is *MINUS*, then ANNA first locates the right child of *MINUS*, finding the compatible mapping (cf. Definition 4) $\mu'_{NB'}$ of $\mu_{NB'}$ in the output of the right child (Lines 11–14). The tuple $(MINUS(\mu'_{NB'}), \mu_{NB'})$ can be added to ψ (Line 15). For instance, consider the first

Algorithm 3 Identifying inappropriate operators

Input: SPARQL query Q, expected mappings $[\![NB']\!]_D$
Output: Explanation ψ
1: Initialization: $\tau(P) = \varnothing$, $OP = \varnothing$, $\psi = \varnothing$;
2: $\tau(P) \leftarrow$ParseQuery (Q); //return a parser tree
3: $OP \leftarrow$PostOrderTraversal$(\tau(P))$;
4: **for all** $op \in OP$ **do**
5: $\quad \Omega_{op} \leftarrow$GetOutputOfOperator$(op)$;
6: $\quad$ **for all** $\mu_{NB'} \in [\![NB']\!]_D$ **do**
7: $\quad\quad$ **for all** $\forall \mu_{op} \in \Omega_{op} : \mu_{op} \not\subseteq \mu_{NB'}$ **do**
8: $\quad\quad\quad$ **if** op is $\sigma(R)$ **then**
9: $\quad\quad\quad\quad \psi$.add$(FILTER(R), \mu_{NB'})$;
10: $\quad\quad\quad$ **end if**
11: $\quad\quad\quad$ **if** op is $\setminus$ **then**
12: $\quad\quad\quad\quad \Omega_r \leftarrow$GetOutputOfOperator$(op \rightarrow rightchild)$;
13: $\quad\quad\quad\quad \mu'_{NB'} \leftarrow$FindCompatibleMapping$(\Omega_r, \mu_{NB'})$;
14: $\quad\quad\quad\quad \psi$.add$((MINUS(\mu'_{NB'}), \mu_{NB'}))$;
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: **end for**
19: **return** ψ.

MINUS of OP in Example 9, ANNA first locates P_2 in $\tau(P)$ and then finds the corresponding $\mu_2 \in [\![P_2]\!]_D : \mu_2 \sim \mu_{NB'}$. The tuple $(MINUS(\mu_2), \mu_{NB'})$ will be added to explanation ψ.

The cost of Algorithm 3 is $O(n \times m \times o)$, where n is the number of operators in $\tau(P)$, m is the size of $[\![NB']\!]_D$, and o is the largest output set for any operation in $\tau(P)$.

6 Experiments

Our proposed explanation model, ANNA, was evaluated in terms of effectiveness and efficiency by performing experiments on two real-world RDF datasets. In particular, we wish to answer i) how well our method compares to the competing techniques?", which is evaluated in Sect. 6.1. Given this, the next question naturally raised would be ii) is the proposed model efficient? does it scale to large knowledge graph?" To investigate this, we explore our model on two real-world large knowledge graphs, which is presented in Sect. 6.2. The detailed experimental settings of our evaluations are described as follows.

Datasets Two RDF datasets, namely English DBpedia and LinkedMDB, were used in the experiments. The DBpedia dataset is the nuclear dataset of Linked Data and contains more than 4.58 million entities and 583 million RDF triples, among which approximately 50,000 entities are typed as films. The LinkedMDB dataset contains 503,000 RDF entities and 6.14 million RDF triples.

Query set It is important to note that, to the best of our knowledge, our work is the first on SPARQL why-not questions. There is no benchmark query set for the evaluation. To address this, we create a SPARQL query set and made it publicly available (see footnote 8 on this page). Fifty SPARQL queries[8] about films were constructed for evaluation. As we analyzed in Sect. 4, SPARQL why-not questions may happen

[8] http://kfm.skyclass.net/anna/queryset.html.

at the triple pattern level or the query operator level, and the query set should include diverse queries to cover all kinds of why-not scenarios. Therefore, the query set has been designed to focus on the following two aspects.

(1) For the triple pattern level, the query set has been designed to include queries with inappropriate direction, and inappropriate RDF terms.
(2) For the query operator level, the query set has been designed to include queries with the operator *AND, OPTIONAL, MINUS, UNION*, different operators blending together, as well as containing *FILTER* conditions.

Six volunteers (graduate students in computer science, but none of them know the details of ANNA) were asked to pose the fifty SPARQL queries independently over LinkedMDB. Volunteers issued why-not questions by specifying missing items that were supposed to be shown up but were not in the final result. Volunteers may issue different why-not questions for the same SPARQL query, as we analyzed in Example 1.

A total of 61 why-not questions ($w1$-$w61$) were obtained from the 42 SPARQL queries, among which 44 why-not questions were caused by inappropriate triple patterns, and the remaining 17 were caused by inappropriate operators.

Baselines In addition, there is no baseline method that can be readily applied; hence, in our experiment, we only compare with the modified versions of an operator-based model based on [7] (denoted by OM). The operator-based model in [7] is often used as the baseline for answering why-not questions in relational databases. The results show that our explanation model significantly outperforms OM. ANNA and OM were written in Java. Apache Jena[9] was utilized as a fundamental toolkit to build ANNA [36]. All experiments were conducted on a quadcore 3.10 GHz PC operating on Windows 7 with 8GB of RAM and a 1TB hard disk.

6.1 Effectiveness of ANNA

6.1.1 ANNA versus OM

By analyzing the explanations for 61 why-not questions, we observed that OM may be not applicable for some cases. ANNA can generate more useful explanations than OM for users to fix original queries. Owing to limitations of space, we selected five typical queries $q_1 \ldots q_5$ from query set to illustrate the evaluation, as shown in Table 2. In order to include all why-not scenarios, they should contain ***FILTER*** (q_1), ***OPTIONAL*** (q_2), ***MINUS*** (q_3), ***UNION*** (q_4) and inappropriate direction (q_5), respectively. The ***AND*** is the basic operator to form basic graph patterns, and it is contained by all $q_1 \ldots q_5$ queries. Tables 3 and 4 show the why-not questions and explanations on $q_1 \ldots q_5$, where $\mu_{w_{ij}}$ denotes the why-not question on query $q_i, i \in [1, 5], j \geq 1$.

Applicability If the absence of the expected mapping occurs at the *MINUS* or *OPTIONAL* graph patterns, then OM is not applicable as opposed to ANNA, such as $\mu_{w_{22}}$, $\mu_{w_{32}}$.

Usefulness (1) Consider $\mu_{w_{11}}$, OM simply identified $FILTER(?date > \text{“1990”})$ as the explanation. However, the explanation computed by ANNA not only points out that $\mu_{w_{11}}$ is excluded because of the condition on $?date$, but also reveals the additional information that the missing film *Batman* was actually released in *1989*.

[9] http://jena.apache.org.

Table 2 Five SPARQL queries $q_1 \ldots q_5$

q_i	Query	Graph pattern
q_1	To find films directed by Tim Burton since 1990. $\pi_{\{?filmname,?date\}}$ $(\sigma_{?date>\text{“}1990''}$ $(P_1 \bowtie P_2))$	P1 = {⟨?film name ?filmname⟩. ⟨ ?film released ?date⟩. ⟨ ?film director ?person⟩} P2 = {⟨ ?person name “Tim Burton″⟩}
q_2	To retrieve the actor and his country if he won Academy Male Best Actor. $\pi_{\{?actorname,\ ?country\}}$ $((P_1 \bowtie P_2) \bowtie$ $(P_3 \bowtie P_4))$	P1 = ⟨ ?actor name ?actorname⟩ P2 = ⟨ ?actor type actor.⟩ P3 = {⟨ ?actor country ?country⟩. ⟨ ?actor gender male⟩} P4 = {⟨ ?actor haswon “Academy Award for Best Actor″⟩}
q_3	To find romantic disaster films released after 2000. $\pi_{\{?filmname,?date\}}$ $(P_1 \setminus (\sigma_{?date>\text{“}2000''}(P_2)))$	P1 = {⟨?film name ?filmname⟩. ⟨ ?film type Romantic films⟩ . ⟨ ? film type Disaster films⟩} P2 = ⟨ ? film released ?date⟩
q_4	To find war or road films released in America. $\pi_{\{?filmname\}}$ $((P_1 \cup P_2) \bowtie P_3)$	P1 = {⟨ ?film type War films⟩} P2 = ⟨ ?film type Road films⟩ P3 = {⟨ ?film name ?filmname⟩. ⟨ ?film country “United States″⟩}
q_5	To find the female directors who directed and acted in the same film, and the film won the Academy Best Picture. $\pi_{\{?directorname,?filmname\}}$ $(P_1 \bowtie P_2 \bowtie P_3$ $\bowtie P_4 \bowtie P_5)$	P1 = ⟨ ?director director ?film⟩ P2 = ⟨ ?director actedin ?film⟩ P3 = ⟨ ?director name ?directorname⟩ P4 = ⟨ ?director gender female⟩ P5 = {⟨ ?film name ?filmname⟩. ⟨ ? film haswon “Academy Award for Best Picture″⟩}

Table 3 Why-not questions on $q_1 \cdots q_5$

$\mu_{w_{ij}}$	Why-not questions
$\mu_{w_{11}}$	$\{?filmname \rightarrow \text{“}Batman''\}$
$\mu_{w_{12}}$	$\{?date \rightarrow \text{“}1995,'' ?filmname \rightarrow \text{“}Batman\ Forever''\}$
$\mu_{w_{21}}$	$\{?actorname \rightarrow \text{“}Spike\ Lee''\}$
$\mu_{w_{22}}$	$\{?actorname \rightarrow \text{“}Leonardo\ DiCaprio,''$ $?country \rightarrow \text{“}United\ States''\}$
$\mu_{w_{31}}$	$\{?filmname \rightarrow \text{“}The\ English\ Patient''\}$
$\mu_{w_{32}}$	$\{?filmname \rightarrow \text{“}Titanic''\}$
$\mu_{w_{41}}$	$\{?filmname \rightarrow \text{“}Atonement''\}$
$\mu_{w_{51}}$	$\{?filmname \rightarrow \text{“}Argo''\}$

With this important information, the user can refine *FILTER* condition to *FILTER* ($?date > \text{“}1989''$), and *Batman* will show up in the query result. (2) Consider $\mu_{w_{12}}$, $\mu_{w_{21}}$, $\mu_{w_{31}}$, $\mu_{w_{41}}$ and $\mu_{w_{51}}$, OM identified the operator AND between two triple patterns as the explanation. However, users are more concerned about which parts of triple patterns failed to retrieve the expected mapping. With the suggested modifications by ANNA, the user can determine why $\mu_{w_{12}}$, $\mu_{w_{21}}$, $\mu_{w_{31}}$, $\mu_{w_{41}}$ and $\mu_{w_{51}}$ are missing.

Table 4 Why-not questions, and explanations generated by ANNA and QM

$\mu_{w_{ij}}$	why-not questions	ANNA	OM
$\mu_{w_{11}}$	$\{?filmname \rightarrow$ "$Batman''\}$	{(FILTER(?*date* > "1990″), {?*film* → *f1333*, ?*date* → "1989,″ ?*person* → p2556, ?*filmname* → "Batman″}}	FILTER
$\mu_{w_{12}}$	$\{?date \rightarrow$ "1995″, $?filmname \rightarrow$ "$Batman\ Forever''\}$	P′ = {⟨ ?film name ?filmname⟩. ⟨ ?film released ?date⟩. ⟨ ?film **producer** ?person⟩. ⟨ ?person name "Tim Burton"⟩}	AND
$\mu_{w_{21}}$	$\{?actorname \rightarrow$ "$Spike\ Lee''\}$	P′ = {⟨ ?actor name ? actorname⟩. ⟨ ?actor type **director**⟩}	AND
$\mu_{w_{22}}$	$\{?actorname \rightarrow$ "$Leonardo\ DiCaprio,''$ $?country \rightarrow$ "$United\ States''\}$	P′ = {⟨ ?actor name ?actorname⟩. ⟨ ?actor type actor⟩. ⟨ ?actor country ?country⟩. ⟨ ?actor gender male⟩. ⟨ ?actor haswon "**Film awards for lead actor**″⟩}	Not applicable
$\mu_{w_{31}}$	$\{?filmname \rightarrow$ "$The\ English\ Patient''\}$	P′ = {⟨ ?film name ?filmname⟩. ⟨ ?film type Romantic films⟩. ⟨ ?film type **War films**⟩}	AND
$\mu_{w_{32}}$	$\{?filmname \rightarrow$ "$Titanic''\}$	{(MINUS({?film → *f25371*, date → "*1997*″}), {?*film* → *f25371*, ?*filmname* → "Titanic″}}	Not applicable
$\mu_{w_{41}}$	$\{?filmname \rightarrow$ "$Atonement''\}$	P′ = {⟨ ?film type War films⟩. ⟨ ?film name ?filmname⟩. ⟨ ?film country "**UK**″⟩} P″ = {⟨ ?film type **Romantic films**⟩. ⟨ ?film name filmname⟩. ⟨ ?film country "**UK**″⟩}	AND
$\mu_{w_{51}}$	$\{?filmname \rightarrow$ "$Argo''\}$	P′ = {⟨ **?film director ?director**⟩. ⟨ ?director actedin ?film⟩. ⟨ ?director name ?directorname⟩. ⟨ ?director gender **male**⟩. ⟨ ?film name ?filmname⟩. ⟨ ?film haswon "Academy Award for Best Actor″⟩}	AND

In summary, ANNA can identify where the expected mapping is missing and generate more informative and useful why-not explanations than OM.

6.1.2 Metrics on the modified graph pattern

For the why-not questions caused by inappropriate triple patterns, the explanations are modified graph patterns. A good modified query should be similar to the original query and have few extra items in the result [18]. We measure the effectiveness of a modified graph pattern by using two metrics.

Similarity metric Given a graph pattern P and its modified graph pattern P', the similarity between P and P' can then be computed as

Table 5 The similarities and the imprecision values of modified graph patterns for $\mu_{w_{12}}$, $\mu_{w_{21}}$, $\mu_{w_{22}}$, $\mu_{w_{31}}$, $\mu_{w_{41}}$ and $\mu_{w_{51}}$

m_w^k	μ_w	*simP(P,P')*	*impP(P,P')*
$m_{w_{12}}^1$	$\mu_{w_{12}}$	0.972	0.143
$m_{w_{21}}^1$	$\mu_{w_{21}}$	0.814	0.382
$m_{w_{22}}^1$	$\mu_{w_{22}}$	0.932	0.193
$m_{w_{31}}^1$	$\mu_{w_{31}}$	0.933	0.167
$m_{w_{41}}^1$	$\mu_{w_{41}}$	0.935	0.212
$m_{w_{41}}^2$	$\mu_{w_{41}}$	0.891	0.25
$m_{w_{51}}^1$	$\mu_{w_{51}}$	0.958	0

$$simP(P, P') = \sum_{i=1}^{n} sim(r_i, r_i')/n \ , \tag{8}$$

where $r_i \in P$, $r_i' \in P'$, r_i and r_i' are RDF terms. $sim(r_i, r_i')$ is computed according to Equation (5), and n is the number of RDF terms in P. The closer to 1 $simP(P, P')$ is, the better P' is.

Imprecision metric The modified graph pattern P' should be as precise as possible in terms of its result. Ideally, $[\![P']\!]_D$ should contain only existing mappings in $[\![P]\!]_D$ and one mapping μ' that satisfies $\mu' \supseteq \mu_w$. For instance, μ' in Example 1 may be $\{?film \rightarrow f1333, ?date \rightarrow$ "1990,"$?person \rightarrow p2556, ?filmname \rightarrow$ "$Batman$"$\}$, which contains $\mu_w = \{?filmname \rightarrow$ "$Batman$"$\}$. Any additional mappings in $[\![P']\!]_D$ are considered irrelevant results that should be minimized. Given a P', let $R \subseteq [\![P']\!]_D$ denote the mappings that contain the why-not question μ_w. We follow [33] and define the imprecision value for P' as the ratio of the number of irrelevant mappings in $[\![P']\!]_D$ to the number of mappings in $[\![P']\!]_D$:

$$impP(P, P') = \frac{|[\![P']\!]_D - R - [\![P]\!]_D|}{|[\![P']\!]_D|}. \tag{9}$$

The closer to 0 $impP(P, P')$ is, the better.

Table 5 reports the similarities and imprecision values of modified graph patterns for $\mu_{w_{12}}$, $\mu_{w_{21}}$, $\mu_{w_{22}}$, $\mu_{w_{31}}$, $\mu_{w_{41}}$ and $\mu_{w_{51}}$, where $m_{w_{ij}}^k$ denotes the modified graph pattern for $\mu_{w_{ij}}$ (*UNION* may lead to more than one modified graph pattern for a why-not question, e.g, $\mu_{w_{41}}$).

For 44 why-not questions caused by inappropriate triple patterns, ANNA generated 48 corresponding modified graph patterns. The average of similarity is approximately 0.91, and the average value of imprecision is approximately 0.17 (11 modified graph patterns imprecision values are 0). Modified graph patterns generated by ANNA have good quality in terms of both similarity and imprecision metrics.

6.2 Efficiency evaluation

6.2.1 ANNA versus OM

In this section, we first measure the execution time required by ANNA to generate explanations on LinkedMDB. Then, we compare the efficiency of ANNA with OM.

Performance of modifying graph patterns We measure the modification time for the 44 why-not questions caused by inappropriate triple patterns. The experimental result shows

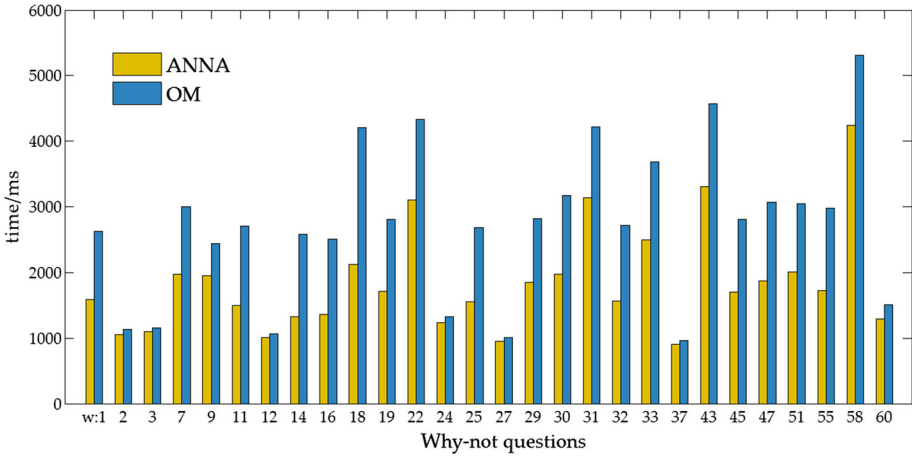


Fig. 21 Time overhead of ANNA and OM

that the modified graph patterns for explaining why-not questions can be generated in less than 6 s, the average time is 3.5 s, and the maximum time is 5.3 s. This amount of time is tolerable considering that users are eventually provided an explanation.

Performance of identifying inappropriate operators We measure the execution time of identifying process for the 17 why-not questions caused by inappropriate operators. The experimental result shows that all inappropriate operators for a why-not question can be identified in less than 4 s, the average time is 1.8 s, and the maximum time is 3.6 s. This amount of time is acceptable for users to obtain explanations.

The capability of ANNA to generate an explanation within a reasonable amount of time is attributed to the divide-and-conquer strategy adopted by ANNA. The query level where a why-not question occurs is first identified by ANNA. Explanations are then generated at the corresponding level. As a result, complex query processing is avoided.

ANNA versus OM Figure 21 reports runtimes for ANNA and OM on 28 out of 61 why-not questions (for the others, OM is not applicable). Among these, 17 why-not questions are caused by inappropriate operators, and others are caused by inappropriate triple patterns. For comparison purposes, we only measure the runtime of inappropriate triple patterns detection and ignoring the time of modification time. We observe that OM is slower than ANNA, because ANNA adopts the BGP partition to identify triple patterns in parallel and detects questionable operators (*MINUS* or *FILTER*) only during the bottom-up traversal.

We then see what occurs when OM is slower than—but comparable to—ANNA, such as *w2, w3, w12, w24, w27* and *w37* in Fig. 21. In OM, the expected RDF terms are pruned out by *AND* between two triple patterns, which is close to the leaf level of the parse tree; hence, the bottom-up traversal of the tree can stop early. ANNA cannot benefit from such an early termination. Although the time complexities of generating these explanations by OM and ANNA are identical, ANNA can retrieve explanations attached with more information to users. (e.g., $\mu_{w_{11}}$ in Sect. 6.1.1).

6.2.2 Sensitivity analysis on different scale datasets

Sensitivity analysis of modifying graph patterns Figure 22 shows the graph pattern modification time for the 44 why-not questions caused by inappropriate triple patterns on

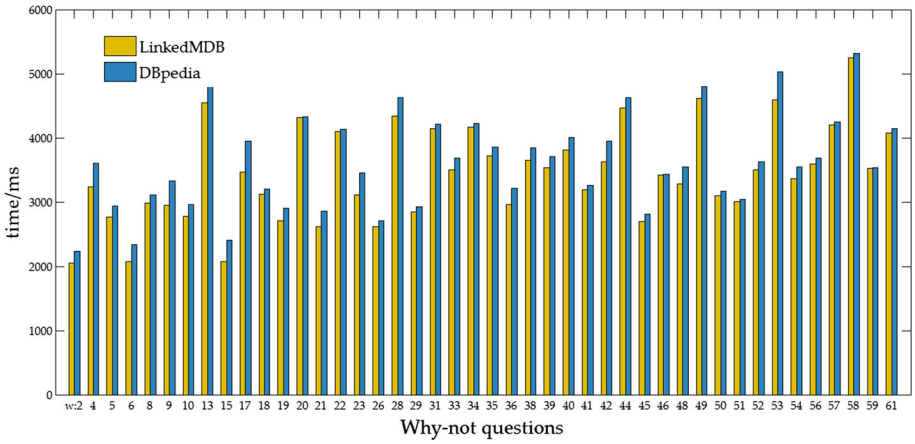


Fig. 22 Time overhead of modifying graph patterns on LinkedMDB and DBpedia

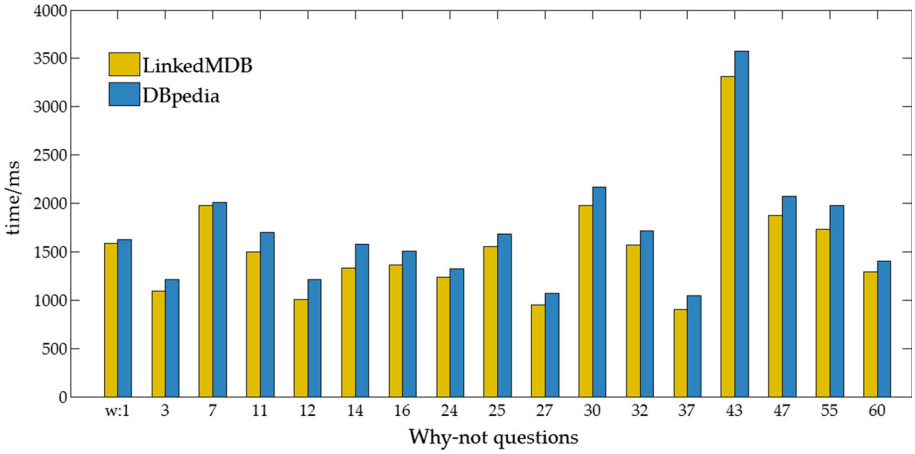


Fig. 23 Time overhead of identifying inappropriate operators on LinkedMDB and DBpedia

LinkedMDB and DBpedia. The average time on DBpedia increased slightly more than that on LinkedMDB, although the size of DBpedia is approximately 90 times larger than that of LinkedMDB. The reason is that modification time depends on the size of the original graph pattern and the ontology of RDF datasets. Two experiments share the same size of the original graph pattern, and the *film* ontologies in LinkedMDB and DBpedia have a similar scale.

Sensitivity analysis of identifying inappropriate operators Figure 23 shows the execution time of identifying process for the 17 why-not questions caused by inappropriate operators on LinkedMDB and DBpedia. The runtimes from LinkedMDB and DBpedia have no significant change, thereby indicating that the scale of the underlying dataset has little effect on the overall runtime of identifying inappropriate operators. At the query operator level, the main factor that influences the efficiency of identifying inappropriate operators is the size of mappings (an input of Algorithm 3), not the entire RDF graphs in datasets.

Figures 22 and 23 show that ANNA is not sensitive to the scale of RDF datasets.

In summary, experimental results of effectiveness show that ANNA is more applicable to SPARQL operators than OM model. And ANNA can also generate modified graph patterns in good quality in terms of both similarity and imprecision metrics. Hence, ANNA can generate high-quality explanations for users. For efficiency, the experimental results show that ANNA can answer why-not questions within a reasonable amount of time, at both triple pattern and query operator levels.

7 Conclusion

In this study, we formalized the problem of answering why-not questions on SPARQL queries for the first time and proposed an explanation model called ANNA. ANNA can generate logical explanations at the triple pattern and query operator levels. Efficient algorithms were designed to implement three modules in ANNA, including computing the necessary BGP, modifying graph patterns and identifying inappropriate operators. ANNA was evaluated on two real-world RDF datasets. Experimental results proved that ANNA could generate high-quality explanations within a reasonable amount of time.

In the future, we will make ANNA to handle exception cases in practice. For instance, a user can be wrong and ask an aberrant why-not question and the explanation is an empty set. Another potential direction for future research is to improve the graph pattern modification method to identify and deliver to the user the most relevant one with the multiplicity of explanation.

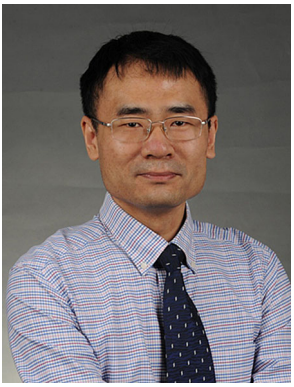
Acknowledgements This work is sponsored by the Fundamental Theory and Applications of Big Data with Knowledge Engineering under the National Key Research and Development Program of China with Grant No. 2016YFB1000903; National Science Foundation of China under Grant Nos. 61721002, 61672419, 61672418, 61532004 and 61532015; MOE Research Center for Online Education Funds under Grant No.2016YB165; Ministry of Education Innovation Research Team No. IRT17R86.

References

1. Baget JF, Benferhat S, Bouraoui Z, Croitoru M, Mugnier ML, Papini O, Rocher S, Tabia K (2016) A general modifier-based framework for inconsistency-tolerant query answering. In: KR, pp 513–516
2. Bhowmick SS, Sun A, Truong BQ (2013) Why not, wine? towards answering why-not questions in social image search. In: Proceedings of the ACMMM. ACM, pp 917–926
3. Bidoit N, Herschel M, Tzompanaki K (2014) Query-based why-not provenance with nedexplain. In: Proceedings of the EDBT
4. Bienvenu M, Bourgaux C, Goasdoué F (2016) Explaining inconsistency-tolerant query answering over description logic knowledge bases. In: AAAI, pp 900–906
5. Bienvenu M, Rosati R (2013) Tractable approximations of consistent query answering for robust ontology-based data access. In: IJCAI, pp 775–781
6. Calvanese D, Ortiz M, Simkus M, Stefanoni G (2013) Reasoning about explanations for negative query answers in dl-lite. J Artif Intell Res 48:635–669
7. Chapman A, Jagadish H (2009) Why not? In: Proceedings of the ACM SIGMOD. ACM, pp 523–534
8. Chen L, Lin X, Hu H, Jensen CS, Xu J (2015) Answering why-not questions on spatial keyword top-k queries. In: Proceedings of the ICDE. IEEE, pp 279–290
9. Cui Y, Widom J (2003) Lineage tracing for general data warehouse transformations. VLDB J 12(1):41–58
10. Damásio CV, Analyti A, Antoniou G (2012) Provenance for sparql queries. In: Proceedings of the ISWC. Springer, pp 625–640
11. Dividino R, Sizov S, Staab S, Schueler B (2009) Querying for provenance, trust, uncertainty and other meta knowledge in RDF. Web Semant Sci Serv Agents World Wide Web 7(3):204–219
12. Eiter T, Fink M, Schüller P, Weinzierl A (2014) Finding explanations of inconsistency in multi-context systems. Artif Intell 216:233–274

13. Elbassuoni S, Ramanath M, Schenkel R, Sydow M, Weikum G (2009) Language-model-based ranking for queries on RDF-graphs. In: Proceedings of the CIKM. ACM, pp 977–986
14. Elbassuoni S, Ramanath M, Weikum G (2011) Query relaxation for entity-relationship search. In: Proceedings of the ESWC. Springer, pp 62–76
15. Gallego MA, Fernández JD, Martínez-Prieto MA, de la Fuente P (2011) An empirical study of real-world sparql queries. In: Proceedings of the USEWOD2011, Hydebarabad, India
16. Gao Y, Liu Q, Chen G, Zheng B, Zhou L (2015) Answering why-not questions on reverse top-k queries. In: Proceedings of the VLDB Endowment, vol. 8. VLDB Endowment, pp 738–749
17. Group, W.C.S.W. (2013) Sparql 1.1 overview. http://www.w3.org/TR/sparql11-overview/
18. He Z, Lo E (2014) Answering why-not questions on top-k queries. IEEE Trans Knowl Data Eng 26(6):1300–1315
19. Herschel M, Hernández MA (2010) Explaining missing answers to spjua queries. In: Proceedings of the VLDB endowment, vol. 3. VLDB Endowment, pp 185–196
20. Huang J, Chen T, Doan A, Naughton JF (2008) On the provenance of non-answers to queries over extracted data. In: Proceedings of the VLDB endowment, vol. 1. VLDB Endowment, pp 736–747
21. Huang H, Liu C, Zhou X (2008) Computing relaxed answers on RDF databases. In: Proceedings of the WISE. Springer, pp 163–175 (2008)
22. Hurtado CA, Poulovassilis A, Wood PT (2006) A relaxed approach to RDF querying. In: Proceedings of the ISWC. Springer, pp 314–328
23. Islam MS, Zhou R, Liu C (2013) On answering why-not questions in reverse skyline queries. In: Proceedings of the ICDE. IEEE, pp 973–984
24. Islam MS, Liu C, Li J (2015) Efficient answering of why-not questions in similar graph matching. IEEE Trans Knowl Data Eng 27(10):2672–2686
25. Kiefer C, Bernstein A, Lee HJ, Klein M, Stocker M (2007) Semantic process retrieval with iSPARQL. In: Proceedings of the ESWC. Springer, pp 609–623
26. Lembo D, Lenzerini M, Rosati R, Ruzzi M, Savo DF (2015) Inconsistency-tolerant query answering in ontology-based data access. Web Semant Sci Serv Agents World Wide Web 33:3–29
27. Miltersen PB, Radhakrishnan J, Wegener I (2005) On converting CNF to DNF. Theor Comput Sci 347(1):325–335
28. Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of sparql. ACM Trans Database Syst 34:1–45
29. Saleem M, Ali MI, Hogan A, Mehmood Q, Ngomo ACN (2015) LSQ: the linked SPARQL queries dataset. In: ISWC, pp 121–131
30. Schmidt M, Meier M, Lausen G (2010) Foundations of SPARQL query optimization. In: Proceedings of the ICDT. ACM, pp 4–33
31. ten Cate B, Civili C, Sherkhonov E, Tan WC (2015) High-level why-not explanations using ontologies. In: Proceedings of the ACM PODS. ACM, pp 31–43
32. Theoharis Y, Fundulaki I, Karvounarakis G, Christophides V (2011) On provenance of queries on semantic web data. IEEE Internet Comput 15(1):31–39
33. Tran QT, Chan CY (2010) How to conquer why-not questions. In: Proceedings of the ACM SIGMOD. ACM, pp 15–26
34. Vidal ME, Ruckhaus E, Lampo T, Martínez A, Sierra J, Polleres A (2010) Efficiently joining group patterns in sparql queries. In: Proceedings of the ESWC. Springer, pp 228–242
35. Wang M, Chen W, Wang S, Liu J, Li X, Stantic B (2017) Answering why-not questions on semantic multimedia queries. Multimed Tools Appl 1–25. https://doi.org/10.1007/11042-017-5151-6
36. Yao S, Liu J, Wang M, Wei B, Chen X (2015) Anna: answering why-not questions for SPARQL. In Proceedings of the ISWC (Demos)
37. Zhang X, Xiao G, Lin Z, Van den Bussche J (2014) Inconsistency-tolerant reasoning with OWL DL. Int J Approx Reason 55(2):557–584
38. Zhou X, Gaugaz J, Balke WT, Nejdl W (2007) Query relaxation using malleable schemas. In: Proceedings of the ACM SIGMOD. ACM, pp 545–556

Meng Wang is currently working toward the PhD degree in computer science at Xi'an Jiaotong University. He received the BS degree in Computing Science from Sichuan University in 2012. His research interests include SPARQL query language and data mining.



Jun Liu is currently a professor in the Department of Computer Science, Xi'an Jiaotong University. He has published more than 70 research papers in various journals and conference proceedings. He received the PhD degrees from Xi'an Jiaotong University, China, in 1995, 1998 and 2004, respectively, all in computer science. His main research interests include text mining, data mining and e-learning.

Bifan Wei is currently an engineer in the Department of Computer Science, Xian Jiaotong University. He received the BS degree in Aerocraft Dynamics Engineering from Beijing University of Aeronautics and Astronautics in 2000, the PhD degree in computer science in 2014 from Xian Jiaotong University, China. His research interests include web data mining, faceted search and taxonomy learning.

Siyu Yao is currently working toward the PhD degree in computer science at Xian Jiaotong University. She received the BS degree in Computing Science from Xian Jiaotong University in 2014. Her research interests include SPARQL query language and data mining.

Honwei Zeng is currently working toward the PhD degree in Computer Science at Xi'an Jiaotong University. He received the BS degree in Computing Science from Harbin Engineering University in 2015. His research interests include natural language processing and dialogue system.

Lei Shi is currently working toward the PhD degree in computer science at Xian Jiaotong University. She received the BS degree in Computing Science from Xian Jiaotong University in 2014. Her research interests include SPARQL query language and data mining.