Data Citation: Giving Credit where Credit is Due

Anonymous
Not given

ABSTRACT

An increasing amount of information is being published in structured databases and retrieved using queries, raising the question of how query results should be cited. Since there are a large number of possible queries over a database, one strategy is to specify citations to a small set of frequent queries – *citation views* – and use these to construct citations to other "general" queries. We present three approaches to implementing citation views and describe alternative policies for the joint, alternate and aggregated use of citation views. Extensive experiments using both synthetic and realistic citation views and queries show the tradeoffs between the approaches in terms of the time to generate citations, as well as the size of the resulting citation. They also show that the choice of policy has a huge effect both on performance and size, leading to useful guidelines for what policies to use and how to specify citation views.

KEYWORDS

Data citation, provenance, scientific databases

ACM Reference Format:
Anonymous . 2017. Data Citation: Giving Credit where Credit is Due. In *Proceedings of 2018 ACM International Conference on Management of Data, SIGMOD Conference 2018 (SIGMOD'18).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An increasing amount of information is being published in structured databases and retrieved using queries, raising the question of how query results should be cited. Typically, database owners specify the citation as a reference to a journal article whose title includes the name of the database and whose author list includes the chief personnel (e.g. the PI, DBA, lead annotator, etc), along with the query and date of access. However, in many cases the content of the query result is contributed by members of the community and curated by experts, who are not on the author list of the journal article. There may also be other "snippets" of information that would be useful to include in the citation that vary from query to query, e.g. descriptive information about the data subset being returned, analogous to the title of a chapter in an edited collection. There is therefore increasing interest in being able to *specify* the content of citations to query results and *automate* the generation of citations [5, 7].

Note that the snippets of information described above play an important *human* role in citation. While the query and date of access (or some form of digital object identifier) is sufficient to locate the query result, it does not give intuition about the content. For example, "Nature, 171,737-738" specifies how to locate the article but doesn't tell you why you might want to do so, whereas adding the information "Watson and Crick: Molecular Structure of Nucleic Acids" does. Furthermore, including information about contributors/curators in the citation to a query result captures the provenance of the data, gives contributors/curators appropriate credit, and encourages their continued contribution.

However, it is infeasible to specify a citation to every possible query result: there are many potential queries over a database, each accessing and generating different subsets of data. Each of these different subsets may be attributable to different sets of people, and have different descriptive information. We must instead find ways of specifying citations for portions of the database which represent *frequent* queries (e.g. web page views), and use these to automatically construct citations for data returned by *general* queries. By *general* queries we mean those submitted to the database by users via web forms or APIs which do not necessarily correspond to frequent queries.

Currently, several databases (e.g. the Reactome Pathway database [1] and eagle-i [2]) enable access to the database via web pages, and specify *in English* what snippets of information are to be included in a citation to the web page. However, they do not automatically generate the citations. In contrast, the IUPHAR/BPS Guide to Pharmacology [3] (GtoPdb) includes citations in the web page results: SQL queries are hard-coded in the web-page form that retrieve the appropriate snippets of information from the database, which are then formatted to create a citation for the web page. Note that these web page views represent frequent queries for which citations are specified. None of these databases provide a citation for *general queries*, although the developers of GtoPdb would like to enable this in the future.

This paper presents a system for generating citations to general queries based on Datalog, building on the idea of *citation views* proposed in [7]. A citation view specifies how the citation is to be constructed for a a particular subset – a view – of the database, and is associated with each frequent query. In our system, when a general query Q is submitted, the views are mapped to the query. Sets of views are then constructed that "cover" Q (*covering sets*). The citations associated with each view in a covering set are *jointly* used to construct a citation to Q. Since there may be more than one covering set for Q, our system also reasons over *alternate* covering sets.

A feature of our system is that the reasoning is performed at the level of *tuples* in the query result, and that each tuple can

[1]http://www.reactome.org/pages/documentation/citing-reactome-publications/
[2]https://www.eagle-i.net/get-involved/for-researchers/citing-an-eagle-i-resource/
[3]http://www.guidetopharmacology.org/

have an individual citation. This is due to the fact that views may have constraints that include only certain tuples or combinations of tuples, as well as that views can be *parameterized*, thereby creating a family of instantiated views [15]. Users of our system may therefore select the *granularity* at which the citation should be constructed, i.e. the subsets of the query result over which an *aggregated* citation should be constructed.

We present three approaches to implementing citation views, in which the reasoning progressively shifts from the tuple level to the schema level, and describe alternative *policies* for the joint, alternate and aggregated use of citation views. Extensive experiments explore the tradeoffs between these approaches as well as the choice of policies in terms of the time to generate citations as well as the size of the resulting citation. Based on these results, we conclude that generating citations for realistic citation views, queries and policies is effective both in terms of running time and citation size, and that the choice between the approaches depends on the granularity with which the DBA wishes citations to be constructed.

Contributions of this paper include:

(1) A semantics for citations to general queries using citation views based on covering sets of mappings between the views and the input query (Section 3.2).
(2) Three approaches to constructing citations for general queries, in which the reasoning progressively shifts from the tuple level to the schema level (Sections 4.1- 4.3). Two of the approaches generate citations to *individual tuples* in the query result, while the last approach generates a citation to the *entire* result. We also describe optimizations that were used in implementing each of the approaches.
(3) Alternative *policies* for the joint, alternate, and aggregated use of citation views, and a description of how the policies are integrated into each of the approaches (Section 4.4).
(4) Extensive experiments using *synthetic* citation views and queries as well as *realistic* citation views and queries for two different choices of policies (Section 5). The experiments show the tradeoffs between the approaches in terms of (i) the time to generate citations as well as (ii) the size of the resulting citation. The experiments show that, for the synthetic case, the choice of policy has a huge effect both on performance and on the size of the generated citations. In particular, when the “best" citation is chosen rather than using “all possible” citations, there is an order of magnitude speedup. The realistic cases show that all three approaches are feasible, although reasoning at the schema level results in a 2-3x performance gain at the expense of generating citations to individual tuples.

As a positive side-effect of the experiments on realistic cases, we have created a dataset that ties computer science publications in DBLP to their NSF funding grants (DBLP-NSF) that will be made available to the community. The dataset (GtoPdb) is already publicly available.[4]

The rest of the paper is organized as follows: Section 2 discusses related work in the digital libraries as well as database communities. The model and running example (GtoPdb) is presented in Section 3, along with a discussion of the relationship of our model to query rewriting using views. Section 4 describes and contrasts the three approaches, and discusses different policies for joint, alternate, and aggregated use of citations. Section 5 presents results of extensive experiments using both synthetic and realistic citation views and policies, and explores the effect of different policies. Practical recommendations to DBAs for constructing citation views and choosing policies can be found in the conclusions (Section 6), along with a discussion of future research. Appendix A contains more details on how the approaches are implemented, and Appendix B contains details on the datasets used in the experiments.

[4]See http://www.guidetopharmacology.org/download.jsp

2 RELATED WORK

Core principles: Two major international initiatives within the digital libraries community have focused on defining core principles for data citation: CODATA [1] and FORCE 11 [8]. In addition to highlighting the idea that data is a research object that should be citable, giving credit to data creators and curators, these principles state a number of criteria that a citation should guarantee, including: 1) *identification and access* to the cited data; 2) *persistence* of data identifiers as well as related metadata (i.e. *fixity*); and 3) *completeness* of the reference, meaning that a data citation should contain all the necessary information to interpret and understand the data even beyond the lifespan of the data they describe.

Computational solutions for data citation often rely on persistent identifiers such as Digital Object Identifiers (DOI), Persistent Uniform Resource Locator (PURL) and the Archival Resource Key (ARK) [13, 18]. While persistent identifiers enable the data to be located and have associated guarantees of persistence (fixity), they do not constitute a full-fledged solution for data citation. In particular, they do not include snippets of information that are useful for human understanding (completeness), nor do they address the issue of how to manage the variable granularity of data to be cited. These and other deficiencies were noted in [5], which posed data citation as a *computational problem*.

Several proposals target the problem of citing **XML** data. The first is a *rule-based citation system* that exploits the hierarchical structure of XML to provide citations to XML elements [6]. The second solution uses the idea of *database views* to define citable units as a key to specifying and generating citations to XML elements [5]. This approach was then extended in [3] to develop a citation generation and dereferencing system for an RDF dataset called eagle-i. The third uses a *machine learning approach* that learns a model from a training set of existing citations to generate citations for previously unseen XML elements [17].

There are three main proposals for citing **RDF** datasets. The first proposes a *nano-publication model* where a single statement (expressed as an RDF triple) is made citable via annotations containing context information such as time, authority and provenance [11]. The model does not specifically address how to cite RDF sub-graphs with variable granularity and the automatic creation of citation snippets. The second defines a methodology based on *named meta-graphs* to cite RDF sub-graphs [16]. Although the approach addresses the variable granularity problem, the snippets of information desired for a citation are not automatically selected. The last proposal is restricted to generating citations for single resources within an RDF dataset [3].

Two approaches deal with citation for **relational databases** and they addresses both identification and fixity. In the first approach [14], a query against the database returns a result set as well as a stable identifier; the identifier includes the version number of the database when queried, and serves as a proxy for the data to be cited. The database is versioned, so that when the stable identifier is later used (dereferenced) the data can be recovered as of the query time rather than the current version.

The second approach [5] proposes a hierarchy of citable units that can be attached to parts of the database. These units can then be used to generate citations for user queries. This idea was formalized in [7], and an architecture and partial implementation were proposed in [4]. Our paper differs from this work by presenting a semantics of *covering sets of mappings* using Datalog, presenting three different approaches to implementing this semantics, showing how policies can be integrated, and presenting a comprehensive experimental analysis of the tradeoffs between the approaches.

3 MODEL

The citation framework is based on *conjunctive queries* [2]. Conjunctive queries are "universal" across different types of databases (e.g. relational, XML, RDF, etc.), and simplify the reasoning used to generate citations.

We start by describing how *citation views* are specified for *frequent* queries and then show how they can be used to generate citations for *general* queries, i.e. queries for which citations have not been specified. We conclude by discussing the connection between citation reasoning and query rewriting using views.

Throughout this section, we will use the GtoPdb database as the running example. In GtoPdb users view information through a hierarchy of web pages: The top level divides information by families of drug targets that reflect typical pharmacological thinking; lower levels divide the families into sub-families and so on down to individual drug targets and drugs. The content of a particular family "landing" page is curated by a committee of experts; a family may also have a "detailed introduction page" which is written by a set of contributors, who are not necessarily the same as the committee of experts for the family. The citations for these views of the database therefore vary.

The citation for GtoPdb as a whole is a traditional paper written by the database owners, a citation to a family page includes the committee members who curated the content for that family page, and a citation to a family detailed introduction page includes the contributors who wrote the introduction for that family.

The simplified GtoPdb schema we will use is (keys are underlined):

Family($\underline{\text{FID}}$, FName, Type)
FamilyIntro($\underline{\text{FID}}$, Text)
Person($\underline{\text{PID}}$, PName, Affiliation)
FC($\underline{\text{FID}}$, $\underline{\text{PID}}$), FID references Family,
PID references Person
FIC ($\underline{\text{FID}}$, $\underline{\text{PID}}$), FID references FamilyIntro,
PID references Person
MetaData(Type, Value)

Intuitively, FC captures the committee members who curate the content of a family page while FIC captures the contributors who author the Family Introduction page of a family. The last table, MetaData, captures other information that may be useful to include in citations, such as the owner of the database ('Owner', 'Tony Harmar'), the URL of the database ('URL', 'guidetopharmacology.org') and the current version number of the database ('Version', '23').

3.1 Citation views

A citation view specifies: 1) the *view definition*; 2) the information to be used to construct the citation (the *citation queries*); and 3) how the information is combined to construct the citation (the *citation function*). The citation function takes the output of the citation queries as input, and outputs a citation in some appropriate format (e.g. human readable, BibTex, RIS or XML). The citation can be thought of as an *annotation* on every tuple in the view result.

To simplify the reasoning for generating citations for general queries, the view definition is a conjunctive query. Although citation queries may be in any language, throughout this presentation we use conjunctive queries. The citation function may also use any language, e.g. Java or Python.

The view definition and citation queries are optionally *parameterized*, where the parameters (*lambda variables*) appear as variables somewhere in the body of the query. [5] A parameterized view creates a set of instantiated views, one for each possible choice of parameters. The number of such views is therefore instance-dependent.

For example, view definitions for the simplified GtoPdb schema could be:

$$\begin{array}{ll}
\lambda F.V1(F,N) & :-Family(F,N,Ty) \\
\lambda F.V2(F,Tx) & :-FamilyIntro(F,Tx) \\
\quad V3(F,N,Ty) & :-Family(F,N,Ty) \\
\lambda Ty.V4(N,Ty) & :-Family(F,N,Ty), F > 60 \\
\lambda Ty.V5(F,N,Ty,Tx) & :-Family(F,N,Ty), \\
 & \quad FamilyIntro(F1,Tx), F = F1
\end{array}$$

All views except $V3$ are parameterized. $V1$ and $V2$ create sets of instantiated views, one for each tuple in Family and FamilyIntro. $V4$ and V5 create sets of instantiated views, one for each type in Family, whereas $V3$ creates one view containing all tuples in Family. Figure 1 shows the effect of views $V1$, $V3$ and $V4$ on a sample instance of Family. For example, $V1$ results in a set of 5 views, $V3$ a single view, and $V4$ a set of 2 views.

We assume that all queries (including view definitions) use fresh variables in every position; any *local* constraints on variables (i.e. those involving a single variable) and *global* constraints (i.e. those involving more than one variable) are expressed as non-relational subgoals of a query.

For each of the views, we define one or more citation queries. Recall that FC captures the committee members who curate the content of a family page while FIC captures the contributors who author the family introduction page:

$$\begin{array}{ll}
\lambda F.\ C_{V1}(F,N,Pn) & :-\ Family(F,N,Ty), FC(F,C), \\
 & \quad Person(C,Pn,A)
\end{array}$$

[5]Also called *binding patterns* in [15].

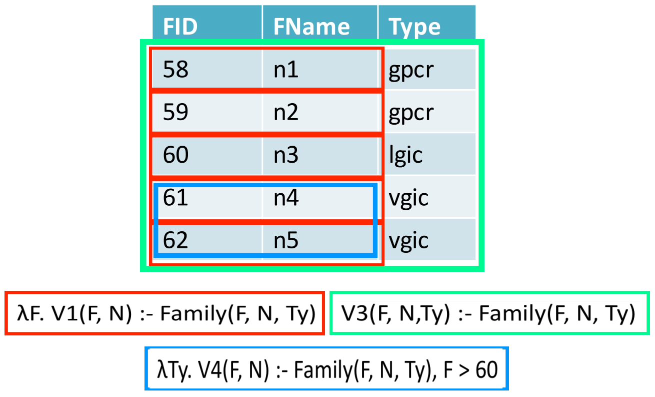

FID
FName
Type
58
n1
gpcr
59
n2
gpcr
60
n3
lgic
61
n4
vgic
62
n5
vgic
λF. V1(F, N) :- Family(F, N, Ty)
V3(F, N,Ty) :- Family(F, N, Ty)
λTy. V4(F, N) :- Family(F, N, Ty), F > 60

Figure 1: Effect of Parameters on Views

$$
\begin{aligned}
\lambda F.\ C_{V2}(F, N, Tx, Pn) &: -\ Family(F, N, Ty),\\
&\quad FamilyIntro(F, Tx),\\
&\quad FIC(F, C), Person(C, Pn, A)\\
C_{V3}(X1, X2) &: -\ MetaData(T1, X1),\\
&\quad T1 = \text{`}Owner\text{'},\\
&\quad MetaData(T2, X2), T2 = \text{`}URL\text{'}\\
\lambda Ty.\ C_{V4}(Ty, N, Pn) &: -\ Family(F, N, Ty), FC(F, C),\\
&\quad Person(C, Pn, A)\\
\lambda Ty.\ C_{V5}(N, Ty, Tx, Pn) &: -\ Family(F, N, Ty),\\
&\quad FamilyIntro(F, Tx),\\
&\quad FIC(F, C), Person(C, Pn, A)
\end{aligned}
$$

The view to which each citation query shown above is associated is given as a subscript, e.g. C_{V1} is associated with $V1$. To ensure that the citation is the same across all tuples in the view, the parameters of the citation query must be a subset of the parameters of the view definition.

The output of the citation functions associated with a view is then used by the citation function to construct a citation. For example, the output of the citation function for V1 parameterized by F=61 (denoted V1(61)) could be:

{ID: '61', Name: 'n4', Committee: ['Hay', 'Poyner']}

We could also associate a citation query with *no* parameters to V1, for example, a citation for the traditional reference paper for GtoPdb as a whole.

3.2 General queries

To give a semantics to citations for *general queries*, we use the following intuition: *If a view tuple can be used to create a tuple in the query result, then the result tuple carries the view tuple's citation annotation.* We formalize this as follows.

Definition 3.1. **Query Extension** Given a query

$$Q(\bar{X}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$$

where $condition(Q)$ are the non-relational subgoals, the *extension of* Q, Q_{ext}, is

$$Q_{ext}(\bar{X}') : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$$

where $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$. Note that $\bar{X} \subseteq \bar{X}'$. A non-relational subgoal is said to be *local* if variables are from the same relational subgoal and *global* otherwise.

Since a view definition is also a query, we use the same notion for V_{ext}.

Definition 3.2. **View Mapping** Given a view definition V and query Q

$$V(\bar{Y}) : -A_1(\bar{Y_1}), A_2(\bar{Y_2}), \ldots, A_k(\bar{Y_k}), condition(V)$$
$$Q(\bar{X}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$$

a **view mapping** M from V to Q is a tuple (h, ϕ) in which:

- h is a partial one-to-one function which 1) maps a relational subgoal A_i in V that uses some variable in $\bar{Y}$ to a relational subgoal B_j in Q with the same relation name; and 2) cannot be extended to include more subgoals of Q.
- ϕ are the variable mappings from $\bar{Y}' = \cup_{i=1}^{k} \bar{Y}_i$ to $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$ induced by h

A relational subgoal B_j of Q is *covered* iff $h(A_i) = B_j$ for some i. A variable x of Q is *covered* iff $\phi(x) = y$ for some y.

Example 3.3. Consider the following query which finds the names of all 'gpcr' families that have an introduction page:

$$Q(N) : -Family(F1, N, Ty), FamilyIntro(F2, Tx),$$
$$Ty = \text{`}gpcr\text{'}, F1 = F2$$

The extension of Q expands the head to include all variables in the body, $Q_{ext}(F1, F2, N, Ty, Tx)$, and $condition(Q) = \{Ty = \text{“}gpcr\text{”}, F1 = F2\}$.

The first condition is local, and the second is global. There are obvious view mappings from each of the views presented above to the body of Q; for example, one possible mapping, say $M1$, maps the first (and only) subgoal of $V1$ to the first subgoal of Q and induces the mapping of variables $\phi(F) = F1,\ \phi(N) = N,\ \phi(Ty) = Ty$.

Note that a view may be in zero or more view mappings for a given query.

Definition 3.4. **Valid View Mapping** Given a database instance D, a view mapping $M = (h, \phi)$ of V is *valid* for a tuple $t \in Q_{ext}(D)$ iff:

- The projection of t on the variables that are mapped in Q_{ext} under the mapping ϕ is a tuple in $V_{ext}(D)$: $\Pi_{\phi(\bar{Y}')} t \in V_{ext}(D)$
- There exists at least one variable $y \in \bar{Y}$ such that $\phi(y)$ is either a distinguished variable or appears in $condition(Q)$.

Given a set of views $\mathcal{V}$, a query Q and a database instance D, we can build a set of valid view mappings $\mathcal{M}(t)$ for each tuple $t \in Q(D)$ according to Definitions 3.2 and 3.4. We then combine different view mappings from $\mathcal{M}(t)$ to create a *covering set* of views for t.

Definition 3.5. **Covering set** Let $C \subseteq \mathcal{M}(t)$ be a set of valid view mappings. Then C is a covering set of view mappings for t iff

- No $V \in \mathcal{M}(t) \setminus C$ can be added to C to cover more subgoals of Q or variables in $\bar{X}$; and
- No $V \in C$ can be removed from C and cover the same subgoals of Q and variables in $\bar{X}$.

Note that for each tuple t there may be a *set* of covering sets, $\{C_1, ..., C_k\}$. In each $C_i = \{M_1, M_2, \ldots, M_l\}$, the citation views are *jointly* used (denoted *) to construct a citation for t, denoted as

$M_1 * M_2 * ... * M_l$. The citations from each C_i are then *alternately* used (denoted $+^R$) to construct a citation for t, denoted as $C_1 +^R \cdots +^R C_p$.

The result of Q is obtained by projecting Q_{ext} over Q's distinguished variables: $Q(D) = \Pi_S Q_{ext}(D)$. Thus a tuple $t \in Q(D)$ may be derived from multiple tuples in $Q_{ext}(D)$. The annotations from all derivations of t are therefore combined to form a citation for t using the abstract operator +, indicating *alternate derivations*. To create the citation for the query result, the annotations of all tuples in the result are combined using the abstract operator *Agg*. The abstract operators *, $+^R$, + and *Agg* are *policies* to be specified by the database owner, and could be union, intersection, the "best" in some ordering over view mappings, or some form of join. We discuss this more in Section 4.4

3.3 Query Rewriting Using Views: Discussion

Query rewriting using views has been used in many data management problems, in particular query optimization and data integration [12]. We now discuss the relationship between covering sets of views in citation reasoning and query rewriting using views.

Query rewriting using views is centered around the notion of query containment: A query $Q1$ is *contained* in a query $Q2$, denoted $Q1 \sqsubseteq Q2$, iff for any database instance D, $Q1(D) \subseteq Q2(D)$. $Q1$ is *equivalent* to $Q2$, denoted $Q1 \equiv Q2$, iff $Q1 \sqsubseteq Q2$ and $Q2 \sqsubseteq Q1$.

In the context of query optimization, the rewriting must be equivalent to the original query. For a given query Q and a set of views $\mathcal{V}$, the goal is to find a subset $\{V_1, V_2, \ldots V_k\} \subseteq \mathcal{V}$ such that $Q' : -V_1, V_2, ...V_k$ and $Q' \equiv Q$. Furthermore, the rewriting should be *optimal* in some sense, for example, in the number of views used or the estimated join cost.

In the context of data integration, the rewriting must be a maximal containment rewriting, which is a weaker condition. For a given query Q and a set of views $\mathcal{V}$, the goal is to find a subset $\{V_1, V_2, \ldots V_k\} \subseteq \mathcal{V}$ such that $Q' : -V_1, V_2, ...V_k$, $Q' \sqsubseteq Q$ and there is no other rewriting Q'' such that $Q' \sqsubseteq Q''$ and $Q'' \sqsubseteq Q$.

For citation reasoning, the query is evaluated on the database; views are virtual. Covering sets of views are then calculated at the level of each tuple in the result, and the reasoning relies on the *provenance* of values. However, reasoning about covering sets of views is similar to reasoning about valid query rewritings in that they are both centered on mappings between subgoals in the views to subgoals of the query. As in data integration, the view mapping may not include all subgoals of the view and may not cover all subgoals of the query.

4 APPROACH

We now describe three approaches to implementing the citation model for general queries discussed in Section 3.2: *tuple level* (TLA), *semi-schema level* (SSLA) and *schema level* (SLA). As the names suggest, an increasing amount of reasoning, in particular that of finding valid view mappings, progressively shifts from the tuple level to the schema level. We close this section by discussing different interpretations of policies, and how they are implemented in each approach. A detailed description of the three approaches can be found in Appendix A.

4.1 Tuple Level

As part of initializing the database to work with the *tuple-level approach* (TLA), the schema of each relation is expanded with a *view vector* column, which identifies all views in which a tuple potentially participates. When a view $V : -B_V$ is added to the database schema, V is added to the view vector of each tuple t occurring in each relation $R \in B_V$ such that t satisfies the *local predicates* for V. Any *global predicates* which compare variables from different relations (e.g. joins) will be checked at query time.

Preprocessing step. When a query $Q : -B_Q$ is evaluated, we first calculate all *possible* view mappings. Some of these mappings may become invalid for individual result tuples depending on whether global constraints for the views hold. In order to enable global predicate checking as well as the evaluation of parameterized views, Q is then extended to include 1) lambda variables under all possible view mappings; 2) view vectors of every base relation occurring in B_Q; and 3) columns representing the truth value of every global predicate under every possible view mapping.

Query execution step. The extended query, Q_{ext1}, is then executed over the database instance D as an SQL query which calculates the truth value of global predicates, yielding an instance $Q_{ext1}(D)$ over which the citation reasoning occurs.

Reasoning step. In first phase of citation reasoning, valid view mappings within each view vector are calculated for each tuple $t \in Q_{ext1}(D)$. A multi-relation view mapping is valid iff all *global predicates* under this mapping are true for t. Invalid view mappings are then removed from the view vectors. In the second phase, combinations of mappings between the resulting view vectors are considered to find the covering sets.

Example 4.1. Given the views provided in Section 3.1, the base relations Family and FamilyIntro are expanded as shown in Tables 1 and 2. Now consider the following query:

$$Q1(FID1, Name, Type, Text) : -Family(FID1, Name, Type), FamilyIntro(FID2, Text), FID1 = FID2$$

All possible view mappings are shown in Table 4. $Q1$ is extended with the global predicate $FID1 = FID2$ (the global predicate in V5 under mapping M5), and the lambda terms shown in Table 3. After deriving valid view mappings from each view vector, the resulting instance of the extended query $Q1_{ext1}(D)$ is shown in Table 5. Note that the lambda terms $FID1$ and $Type$ already appear as distinguished variables in $Q1$ and are therefore not repeated, and that the global predicate $FID1 = FID2$ appears in the body of $Q1$ and is therefore not explicitly evaluated. The final query result with covering sets is shown in Table 6. Parameterized views are instantiated by passing the parameter values (e.g. V1(59) indicates V1 for family_id=59). There are no "+" terms since projection does not change the result; the key, family_id, is retained in the result.

Population step. Deriving covering sets tuple by tuple is time-consuming especially when the query result is very large. However, it is possible to find subsets of tuples that will share the same covering sets using the view vectors and boolean values of global predicates returned in the extended query. By grouping tuples that share the same view vectors and boolean values of global

Table 1: Sample table for base relation Family

Family_id	Name	Type	View vector
58	n1	gpcr	V1,V3,V5
59	n2	gpcr	V1,V3,V5
60	n3	lgic	V1,V3,V5
61	n4	vgic	V1,V3,V4,V5
62	n5	vgic	V1,V3,V4,V5

Table 2: Sample table for base relation FamilyIntro

Family_id	Text	View vector
58	tx1	V2,V5
60	tx2	V2,V5
61	tx3	V2,V5
62	tx4	V2,V5

Table 3: Lambda terms in the view mappings

View mappings	λ terms
M1	$FID1$
M2	$FID1$
M4	$Type$
M5	$Type$

Table 4: All possible view mappings for $Q1$

View	View mapping	h: mappings on relations	ϕ: mappings on variables	Subgoals covered
V1	M1	$Family \rightarrow Family$	$F \rightarrow FID1$, $N \rightarrow Name$, $Ty \rightarrow Type$	Family
V2	M2	$FamilyIntro \rightarrow FamilyIntro$	$F \rightarrow FID2$, $Tx \rightarrow Text$	FamilyIntro
V3	M3	$Family \rightarrow Family$	$F \rightarrow FID1$, $N \rightarrow Name$, $Ty \rightarrow Type$	Family
V4	M4	$Family \rightarrow Family$	$F \rightarrow FID1$, $N \rightarrow Name$, $Ty \rightarrow Type$	Family
V5	M5	$Family \rightarrow Family$, $FamilyIntro \rightarrow FamilyIntro$	$F \rightarrow FID1$, $N \rightarrow Name$, $Ty \rightarrow Type$, $F1 \rightarrow FID2$, $Tx \rightarrow Text$	Family FamilyIntro

Table 5: Result of executing the extended query, $Q1_{ext1}(D)$

FID1	Name	Type	Text	Valid view mappings from view vector 1	Valid view mappings from view vector 2
58	n1	gpcr	$tx1$	M1,M3,M5	M2,M5
60	n3	gpcr	$tx2$	M1,M3,M5	M2,M5
61	n4	vgic	$tx3$	M1,M3,M4,M5	M2,M5
62	n5	vgic	$tx4$	M1,M3,M4,M5	M2,M5

Table 6: The final result, $Q1(D)$, annotated with the covering sets

FID1	Name	Type	Text	Covering sets
58	n1	gpcr	$tx1$	M3*M2(58) $+^{R}$M5('gpcr')
60	n3	lgic	$tx2$	M3*M2(60) $+^{R}$ M5('lgic')
61	n4	vgic	$tx3$	M3*M2(61) $+^{R}$ M5('vgic') $+^{R}$ M1(61)*M4('vgic')*M2(61)
62	n5	vgic	$tx4$	M3*M2(62) $+^{R}$ M5('vgic') $+^{R}$ M1(62)*M4('vgic')*M2(62)

predicates, deriving covering sets can be done only once per group and then propagated to all tuples within the group. For example, in Table 5, the first two tuples form one group and the third and fourth tuples form another group. This optimization leads to significant performance gains.

Aggregation step. We find the covering sets for the entire query result (or some subset of the query result) by taking the *Agg* of the covering sets of the selected tuples.

Citation generation step. The citation is then calculated by evaluating the citation queries and functions, and will be discussed in Section 4.4. Note that although the body of $Q1$ is the same as $V5$, the citation associated with $V5$ may not be the best choice for the final query result, since $V5$ is *parameterized* by Type. This would lead to a set of associated citations, one for each instance of $V5$, whose cardinality would be the number of different types.

Discussion. While the reasoning used in TLA may seem unnecessarily complex for the running example, it is necessary to handle the general case. For example, if the input query was the product of Family and FamilyIntro, M1-M5 would still be possible view mappings using TLA. However, the validity of M5 for a single tuple depends on whether the join condition in M5 is met, since the join condition is missing in the input query and may not hold for all the tuples in the query result. A view may also be involved in more than one view mapping.

Note that the final step of this approach – generating citations – is delayed until the user selects the tuples of interest (or the entire query). This is due to the fact that executing the citation queries tuple by tuple can be time consuming.

4.2 Semi-Schema Level

The *semi-schema-level approach* (SSLA) does not extend the schema of base relations. Instead, the extended query explicitly tests for both *global* and *local predicates*. Since many of the steps are the same as for TLA (e.g. aggregation and citation generation), we focus on those that differ.

Preprocessing step. As before, when a user query $Q : -B_Q$ is submitted, all the possible view mappings are calculated. The query is then extended to include 1) lambda variables under all the possible view mappings; and 2) columns representing the truth value of every global and local predicates. Since base relations are not annotated, no view vectors are returned. The extended query, Q_{ext2}, is then executed on the database yielding an instance $Q_{ext2}(D)$.

Reasoning step. In the first phase, the valid view mappings for each tuple $t \in Q_{ext2}(D)$ are derived based on the truth values of the *global* and *local predicates* (all must be true). In the second phase, a set of valid view mappings $VM(t)$ are derived for each tuple t. The covering sets for t are then determined within $VM(t)$.

Example 4.2. We return to Example 4.1, and show the result of the extended query $Q1_{ext2}(D)$ in Table 7. As before, the lambda terms all appear as distinguished variables and the global predicate appears in the body of $Q1$, so the only additional information is the local predicate test for V4 under mapping M4, $FID1 > 60$. The

Table 7: The instance of the extended query $Q_{ext2}(D)$

FID1	Name	Type	Text	$FID1 > 60$
58	n1	gpcr	$tx1$	False
60	n3	lgic	$tx2$	False
61	n4	vgic	$tx3$	True
62	n5	vgic	$tx4$	True

result of this test shows that V4 is only valid for the last two tuples, while the view mappings of the other four views are valid in all four tuples. The final query result with covering sets is the same as for TLA and shown in Table 6.

Discussion. While TLA and SSLA will always produce the same result, there are two salient differences which will have performance implications: First, the schema of base relations is not extended in SSLA and therefore less space is used. Second, the extended query in TLA includes the truth value of global predicates as well as the view vectors (which grow with the number of views) whereas the extended query in SSLA includes the truth value of all local and global predicates.

4.3 Schema Level

The *schema level approach* (SLA) does all reasoning at the level of the database schema (including key and foreign key constraints), view definitions and the input query. Therefore, it is *instance independent* and finds a group of covering sets that is valid for all possible instances of the database. The SLA implementation borrows some ideas from query rewriting using views techniques proposed in [9].

Reasoning step. The first phase of reasoning in SLA calculates the valid views. Since the reasoning must be instance independent, a view mapping M is said to be a *valid view mapping* for Q iff it is a valid view mapping for *every* tuple $t \in Q_{ext}(D)$ for *every* instance D (c.f. Definition 3.4). The algorithm therefore reasons over the global and local predicates of V and Q to determine whether the non-relational subgoals of V that are mapped to Q imply the non-relational subgoals of Q involving the mapped variables.[6] This involves checking that all global predicates in V involving mapped variables are also in Q, and checking that the local predicates of V involving mapped variables are less restrictive than Q. It also checks whether relational subgoals in V that are *not* mapped to a subgoal in Q restrict the result by examining key-foreign key relationships.

Covering sets for Q are then calculated from the set of valid views. The algorithm uses a number of clever optimizations (e.g. indices over the views as well as the query, see Appendix A) to prune the search space and speed up matches between sets of views and Q.

Example 4.3. Returning to Example 4.1, using schema level reasoning M4 would not be considered since the constraint $F > 60$ does not appear in Q and is therefore more restrictive. Note that the constraint does not hold for all tuples in all possible instances, including the one shown in Table 1. However, M5 *would* be considered since it includes all relational subgoals in M5, and the (mapped)

[6]Recall that not all relational subgoals in V may be mapped to a subgoal in Q.

Table 8: Interpretations of the abstract operators

Operator	Interpretation
$*$	join, union
$+^R$	union, min
$+$	union
Agg	intersection, union

global constraint in M5 is also in Q. Hence the resulting covering set would be M3*M2 $+^R$M5.

Now suppose the query were modified to:

$$Q2(FID, Name, Type) : -Family(FID, Name, Type), FID > 70$$

The local predicate of M4, $FID > 60$, is less restrictive than the corresponding local predicate of $Q2$, $FID > 70$. Hence M4 is a valid view mapping for $Q2$. However there is now no valid mapping from M5 to $Q2$, since tuples in Family are restricted by the join with FamilyIntro – there is a foreign key constraint from FamilyIntro to Family, but not vice versa.

Query execution step. In order to evaluate citations for parameterized views that appear in the resulting covering sets, the query Q must be extended to include all view parameters that do not appear as distinguished variables. The extended query, Q_{ext3}, is then evaluated; view parameters are used to construct the final citations, and the query result $Q(D)$ is obtained by projecting $Q_{ext3}(D)$ over the distinguished variables of Q.

Discussion. Similar to SSLA, SLA does not require the schema of base relations to be extended. Although all three approaches require the query to be extended prior to evaluation, SLA only extends with the necessary view parameters, whereas TLA additionally extends the query with global conditions and SSLA extends with both global and local conditions. Most significantly, SLA does not reason over individual tuples which leads to considerable performance gains.

However, SLA does not generate a *per tuple* citation which is useful if users wish to cite subsets of the query result; it also may not generate the "most specific" citation if the instance of query result happened to satisfy local predicates in a view.

4.4 Generating Citations

The output of the approaches described above is an annotation of covering sets on the query result as a whole (in the case of SLA), or on each tuple in the query result (in the case of TLA/SSLA). Annotations on tuples are then combined using *Agg* in TLA/SSLA to create an annotation for the query result (or a subset of the query result). We now discuss how citations are constructed from covering set annotations in each of these approaches.

The interpretations of the abstract operators $*$, $+^R$, $+$ and *Agg* explored in this paper are shown in Table 8. The interpretations of the last three ($+^R$, $+$, and *Agg*) are implemented at level of *covering sets*, whereas that of $*$ is implemented at the level of *citations* (which in our case are JSON objects).

The first step is to evaluate $+^R$, which has two interpretations: *union* and *min*. The *union* of covering sets is straightforward, although it can lead to very large citations. In contrast, the goal of *min* is to find the covering set with minimum cost (according to some custom cost function), and it is evaluated as the covering sets are being constructed. It therefore has the advantage of avoiding enumerating all covering sets, thereby significantly reducing the cost of this step in all three approaches. Note that the problem of finding a min-cost covering set can be formalized as a set cover problem, which is NP-complete. However, a greedy algorithm can be applied to derive an $O(logn)$-approximate solution [19]. (See the Appendix for the details of the cost functions and the greedy algorithms.)

Intuitively, the cost function we use chooses the covering set with the smallest number of views in the $*$-term, balanced by the number of *unmatched* terms, including unmatched subgoals, unmatched distinguished variables, and unmatched lambda terms in each view. For example, in $Q1$ the parameters Family_id and Type are not equated to constants and therefore M1, M2, M4 and M5 all have unmatched lambda terms. When they appear in a covering set, this leads to an enumeration over all instantiated views in the result set.

Example 4.4. Returning to $Q1(D)$ in Table 6, assume that the interpretation of $+^R$ is *union*. For the first tuple in the table, the result would be {M3*M2(58), M5('gpcr')}.[7]

Now assume that the interpretation is *min*. Since both terms contain a parameterized view with unmatched lambda terms (which is expensive), the term with the fewer views is chosen and the result would be {M5('gpcr')}.

The result of evaluating $+^R$ is a set of covering sets (a unary set in the case of *min*). The second step is to evaluate + by taking the union over these sets for tuples that are unified in the projected result. In our running example, there are no + terms.

The third step evaluates *Agg*, which aggregates covering sets to generate covering sets for the query result (or for subsets of the query result). When *union* is used, all covering sets appearing for some tuple become part of the final result; when *intersection* is used, only the covering sets appearing across all selected tuples become part of the final result. Note that in *intersection* the lambda terms are ignored; for example, M5('gpcr') and M5('vgic') are both considered to be instances of M5. Thus if a view is parameterized and appears in the covering sets to be aggregated, the union of all mapped instances of the view will be used.

Example 4.5. Assuming the interpretation of *Agg* is *intersection* and the interpretation of $+^R$ is *min*, the result across all four tuples is {M5}. Since M5 is parameterized by Ty and the associated JSON-formatted citations were those in Table 9, the final annotation becomes {M5('gpcr'), M5('lgic'), M5('vgic')}, and the citation will be:
{author: ['Jens', 'Rodrigo', 'John'], Committee: ['Andrew', 'Leo', 'Joel'], Type: ['gpcr', 'vgic', 'lgic']}

If, however, just the first tuple was selected and *union* was used for $+^R$, the resulting annotation would be: {M3*M2(58), M5('gpcr')}.

After evaluating $+^R$, +, and *Agg*, we are left with a set of $*$ expressions, which are implemented at the level of the citations. Thus, the $*$-operator takes as input the citations of its operands, which in our implementation are JSON objects, and returns their union or join (depending on the interpretation).

[7]We do not reason about the expected number of instantiated views based on key versus non-key attributes, or the size of underlying domains.

Table 9: Citations for sample view mappings

View	Result of citation function
M2(58)	{ID: '58', author: ['Mark'], Committee: ['Hay', 'Poyner']}
M3	{author: ['Steve', 'Roger'], Committee: ['Hay', 'Justo']}
M5('gpcr')	{author: ['Jens'], Committee: ['Andrew'], Type: 'gpcr'}
M5('lgic')	{author: ['Rodrigo'], Committee: ['Leo'], Type: 'lgic'}
M5('vgic')	{author: ['John'], Committee: ['Joel'], Type: 'vgic'}

Example 4.6. Suppose the resulting annotation was {M3*M2(58), M5('gpcr')}. If the interpretation of $*$ is *join*, the citation for covering set M2(58)*M3 will become the single object:
{ID: '58', author: ['Mark', 'Steve', 'Roger'], Committee: ['Hay', 'Poyner', 'Justo']}.
If the interpretation is $Union$, the citation will be the set of objects: {{ID: '58', author: ['Mark'], Committee: ['Hay', 'Poyner']}, {author: ['Steve', 'Roger'], Committee: ['Hay', 'Justo']}}.

5 EVALUATION

5.1 Experimental design

We implemented all three approaches in Java 8 and used PostgreSQL 9.6.3 as the underlying DBMS. All experiments were conducted on a linux server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz and 64GB of central memory.

Datasets. Our experiments used two datasets. The first is the GtoPdb database described in Section 3. This database has information about 8978 chemical structures (ligands) and the 2825 human targets they act on. Each target (and ligand) has citation information, such as contributors and/or curators, associated with it.

We also developed a second database that connects computer science publications—extracted from the DBLP datasets— to their NSF funding grants—extracted from the National Science Foundation grant dataset. We will refer to this database as DBLP-NSF. The idea was to add funding information to traditional paper citations, and to be able to vary between citing the conference proceedings (if large numbers of papers from the same conference were in the result set) and citing individual papers. DBLP-NSF consists of 17 relations (authors, papers, conferences, grants, etc.). Author is the largest relation with about six million tuples, and the average size across all relations is about 0.6 million tuples.

Workloads. We evaluated our approaches on two types of workloads: *synthetic* and *realistic*. To test our approaches under large workloads, we created large query results by executing synthetic queries on the GtoPdb dataset. Queries were built using a *user query generator* which takes as input 1) the number of subgoals; 2) the number of distinguished variables in the query; and 3) the number of tuples in the query result. We also implemented a *view generator* which takes as input: 1) the number of views; 2) the number of lambda terms in total; and 3) the total number of predicates. Each generated view has a single citation query attached to it. In the experiments, the configurations of the query generator and view generator ensure that there is only one view mapping for each view.

For realistic workloads, we used citation views and general queries from anticipated workloads of GtoPdb and DBLP-NSF. For

Table 10: Notation used in the experiments

Notation	Meaning
full case	$(*, +^R, +, Agg) = (join, union, union, union)$
min case	$(*, +^R, +, Agg) = (join, min, union, intersection)$
t_{cs}	time to derive covering sets for the entire query
t_{pre}	time for preprocessing step in TLA and SSLA
t_{qe}	time to execute extended query in TLA and SSLA
t_{re}	time for reasoning step in TLA, SSLA and SLA
t_{pop}	time for population step in TLA and SSLA
t_{agg}	time for aggregation step in TLA and SSLA
t_q	query time in SLA
t_{cg}	time for citation generation step in the three approaches
N_{cs}	number of covering sets for the entire query
N_v	number of view mapping to query
N_{iv}	number of instantiated views
N_p	number of predicates under all the view mappings
N_l	number of lambda terms under all the view mappings
N_t	number of tuples in the extended query result

GtoPdb, general queries were derived by consulting with the database owners, and views were designed based on its web-page views. For each view, the corresponding citation query is the query used to generate the hard-coded citations on the web-page. For DBLP-NSF, citation views were designed to correspond to citations to single paper, single conference and single grant. General user queries simulate cases where users are interested in papers from certain authors, certain conferences and certain years together with the grant information of those papers. See Appendix B for details.

As discussed in Section 4.4, there are several different interpretations of joint, alternative. and aggregated policies. We focus on two interpretations:
$(*, +^R, +, Agg) = (join, union, union, union)$, called the *full* case, and
$(*, +^R, +, Agg) = (join, min, union, intersection)$, called the *min* case.

The goal of our evaluation is to study the size of citations and the time performance of the three approaches under different policies and workloads.

In terms of the time performance, an important metric to consider is the time to derive covering sets for the query (t_{cs}) in each of the three approaches. As discussed in Section 4, TLA and SSLA compute covering sets for the query in five steps: preprocessing, query execution, reasoning, population and aggregation. Each step has a time overhead (t_{pre}, t_{qe}, t_{re}, t_{pop} and t_{agg} respectively), and under different policies and workloads the major overhead may come from a different step. Unlike TLA and SSLA, SLA only has two steps: reasoning (t_{re}) and query execution (t_q). We also provide an incremental analysis of the time to derive the covering sets. After the covering sets are derived using either of the three approaches, the citation generation step produces a formatted citation, the time for which is denotes by t_{cg}.

To evaluate the citation size, we measure the number of covering sets (N_{cs}) for the entire query. View parameters also influence the citation size. For example, the citation size of non-parameterized views is smaller than that of parameterized views, since each parameter value in the result generates a different instantiated view and thus a different citation. Furthermore, more instantiated views potentially increase the citation generation time (t_{cg}). Consequently, we also measure the total number of instantiated views (N_{iv}) in the covering sets and explore the correlations between N_{iv} and t_{cg}.

Table 10 provides a summary of the policies, timing and size notations.

Our evaluation studies various factors that affect the time to generate citations and their size, and addresses the following evaluation questions (EQs):

- EQ1: How do the different policies influence performance and citation size?
- EQ2: What is the effect of N_v, N_p and N_l on the time performance and citation size?
- EQ3: What is the scalability of our approaches? That is, how the size of extended query result (denoted N_t) influence the time performance?
- EQ4: What is the performance and citation size of the three approaches in realistic scenarios?

Now we describe the experiments performed to answer these questions.

Exp1. The first experiment evaluates the effect of N_v on time performance and citation size. We configured the query generator to generate a query Q that covered four randomly selected relations in its body, and produced a result set of about one million tuples ($N_t = 10^6$) using subset of the product of those four relations. The view generator was iteratively executed 40 times. At each iteration, one more view is added, in which the relations and the distinguished variables all come from Q so as to ensure valid view mappings. In this experiment, we do not consider other view features such as lambda terms and predicates.

Exp2. This experiment tests how N_p influences the performance and size of citations. Like Exp1, the query generator randomly picked four relations and ensured $N_t = 10^6$. However, the view generator fixed the number of views and varied N_p. The view generator was executed 40 times. Initially, we randomly generated 15 views (with no lambda terms or predicates) with valid view mappings to the query Q. At each iteration, we added randomly generated predicates to create more groups.

Exp3. This experiment evaluates the effect of N_l on time performance and citation size. The same settings as in Exp1 are applied to the query generator. N_v is fixed to 15 and the view generator adds one lambda terms to some view in each iteration and up to 50 iterations are executed.

Exp4. This experiment evaluates the scalability of our approaches in terms of time performance, so N_t is varied. The view generator randomly generates 15 views (i.e. $N_v = 15$) with randomly assigned local predicates and lambda terms. The query generator randomly generates a query with four relations but varies the result size (from 10^2 to 10^7) at each iteration.

Exp5. This experiment evaluates how well the proposed approaches handle realistic workloads in the GtoPdb dataset. In this experiment, 14 views were created and each view has one associated citation query according to the web-page views. Eight user queries were collected from the owners of GtoPdb.

Exp6. This experiment is conducted on the DBLP-NSF dataset. Six views are used, each of which is associated with 1-2 citation queries. Three typical user queries are used as input. The first ($q1$) asks for the titles of papers in a certain conference (e.g. VLDB), while the second (q2) retrieves the titles of all papers published by a given author in a given year. These correspond to searches over the DBLP dataset where users are interested in papers of specific authors or conferences. The third (q3) returns the NSF grants that support papers in a given conference (e.g. VLDB).

5.2 Synthetic workloads

In this subsection, experimental results under synthetic workloads (Exp1-4) are reported and analyzed.

Exp1. For the *full* case, results show that thousands of covering sets are generated as the number of view mappings exceeds 30 (Figure 2a), and that (as expected) the corresponding time to generate them t_{cs} increases exponentially (Figure 2b). As shown in Figure 2b, the reasoning time t_{re} is the major overhead when N_v is over 25, leading to convergence of the three approaches in terms of t_{cs}.

Figure 2c shows the results for the *min* case, which has a huge speed-up compared to the *full* case. Notice that t_{cs} is steady even with a large N_v, and that in the worst case t_{cs} is acceptable (about 25 seconds).

The experimental results partially answer EQ2—what is the effect of N_v on the time performance and citation size. In the *full* case, exponentially large covering sets are generated, taking up to 10 minutes as N_v becomes large. However, since each covering set represents a possible citation, it is unreasonable to generate thousands of citations. On the other hand, the *min* case returns the "best" citation, which reduces t_{re} to a few milliseconds and leads to a steady t_{cs} as N_v increases.

The performance difference between the three approaches is also worth noting. In the *min* case, SLA is faster than the other two, which is expected as it only reasons over schemas. For TLA and SSLA, in the population step, we need to execute extra steps to build a mapping between the each tuple and the covering sets so that the citations in the tuple level are available.

Exp2. In theory, in SSLA and TLA, the number of predicates N_p can influence the time performance in three ways. First, more predicates add more complexity to the extended query and thus increases the time to execute the query t_{qe}. Second, it can potentially create more groups in the extended query result, incurring more reasoning time t_{re}. Third, when *Agg* is interpreted as union, more groups means more covering sets in the schema level and thus more citation generation time t_{cg}. However, N_p has no effect on time performance and citation size in SLA since the predicates are not used for extending the query and no grouping or aggregation is involved.

Figures 3a and 3b show how t_{cs} and its major timing components (t_{qe} and t_{re}) are influenced by N_p in the *min* and *full* cases, respectively. Figure 3c shows the trend of t_{cg} and N_{cs} (they are the same for the three approaches) as N_p increases in the *full* case.

In Figure 3a, t_{qe} is also included for TLA and SSLA, which shows that in the *min* case, increasing N_p results in large increases in the (extended) query execution time for SSLA. However, N_p has no influence on TLA or SLA. The same is true for the *full* case (Figure 3b).

For the *full* case, the reasoning time t_{re} becomes a major overhead as N_p increases for both TLA and SSLA. In terms of citation size, the number of covering sets N_{cs} grows with N_p, which also drives citation generation time t_{cg} to exceed one minute with a large N_p. Thus EQ2 is partially answered, i.e. what's the effect of N_p on the time performance and citation size.

The poor performance of SSLA for t_{qe} is due to the complexity of the extended query. Recall that the boolean values of local predicates are explicitly evaluated and then used for grouping in SSLA, which is is not necessary in TLA (Section 4). Since the number of annotation columns in TLA is the number of relations in the query, the number of attributes used for grouping is not affected by N_p. However, more local predicates bring in more columns to be involved in the grouping process and thus leads to the fast growth of t_{qe}. Besides, more predicates can create more groups in the query result, which incurs more t_{re} in total. Unlike TLA and SSLA, SLA won't be influenced by the predicates at all since the reasoning is at the schema level without any grouping process in the query. In terms of citation size, since in *full* case, *Agg* is interpreted as union and the number of covering sets is fixed in each group, larger number of groups leads to numerous covering sets derived for the entire query in TLA and SSLA. Notice that in *min* case, *Agg* is intersection, thus at most one covering set is generated regardless of the number of groups. We therefore omit those results.

Exp3. Theoretically, in all three approaches, when there are more lambda terms, more attributes must be returned in the extended query in order to evaluate citations for the parameterized views. This should increase the query execution time t_{qe}. However, Figure 4 shows that the number of lambda terms N_l has almost no effect on the overall performance of the derivation process t_{cs}, of which t_{qe} is a component. The *full* case shows similar results (ignored here). This again addresses EQ2.

In this experiment, N_{cs} is fixed since N_v is a constant and $N_p = 0$. Note, however, in *full* case, more lambda terms introduces more (fine-grained) views (N_{iv}) and leads to a longer citation generation time (t_{cg}) for the more lambda values, as shown in Figure 5 (N_{iv} and t_{cg} don't vary too much in *min* case, not shown here). It shows the effect of N_l on the citation size, which is part of solutions of EQ2.

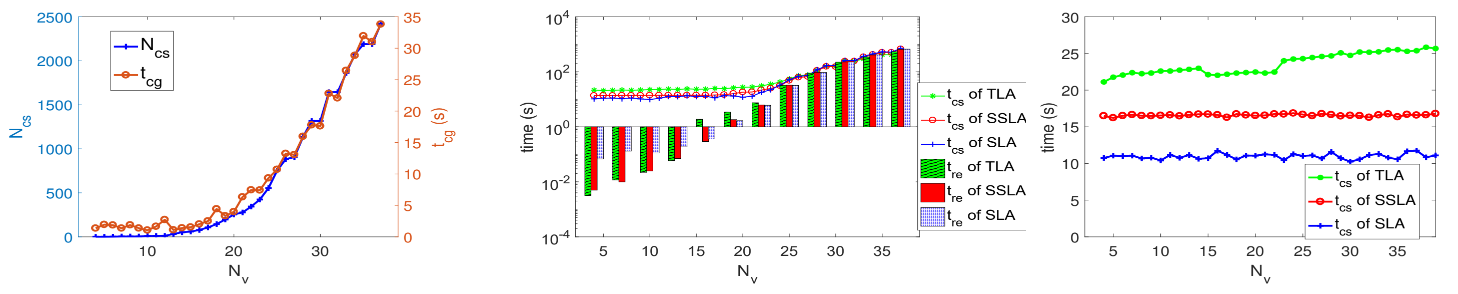
Exp4. Figure 6 shows that when there are fewer than 10^7 tuples in the query result, the time to calculate covering sets t_{cs} in all three approaches is less than 200 seconds, and that of of SLA is less than 60 seconds, which shows the scalability of our approaches (EQ3).

Discussion. These experimental results address EQ1, EQ2 and EQ3. For EQ1, the *min* case and the *full* case mainly differ in the reasoning step, which leads to large differences in performance. In the *min* case, even in complicated scenarios, the reasoning time t_{re} is small (a few milliseconds) since the search space is pruned. In contrast, in the *full* case, the three approaches is very slow in reasoning step since all possible citations are generated.

5.3 Realistic workloads

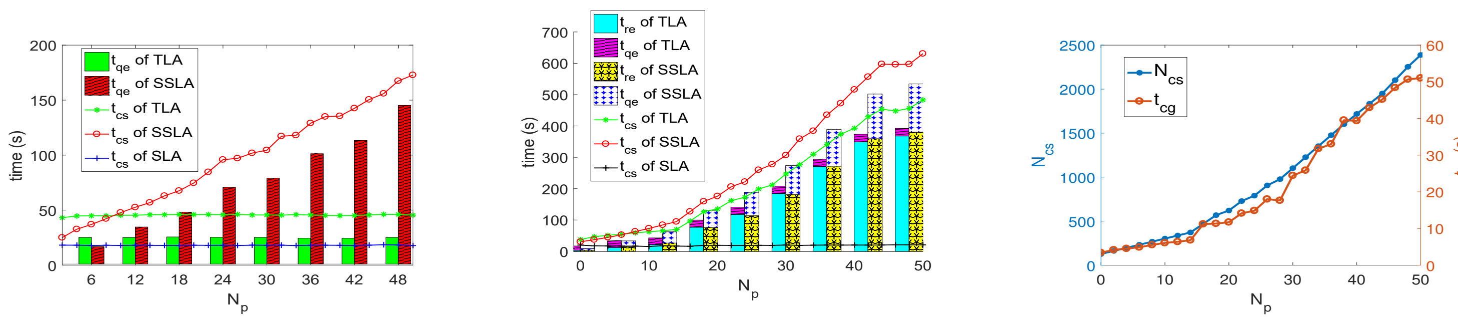
Table 11 shows experimental results for both Exp5 and Exp6.

Exp5. The time to generate covering sets t_{cs} and the time for the citation generation step t_{cg} for all the queries is very small (less



(a) N_{cs} and t_{cg} in *full* case (b) t_{cs} and t_{re} in *full* case (log scale in Y-axis) (c) t_{cs} in *min* case

Figure 2: time performance and citation size VS number of view mappings



(a) t_{cs} and t_{qe} in *min* case (log scale in Y-axis) (b) t_{cs}, t_{qe} and t_{re} in *full* case (log scale in Y-axis) (c) N_{cs} and t_{cg} in *full* case

Figure 3: time performance and citation size VS number of predicates

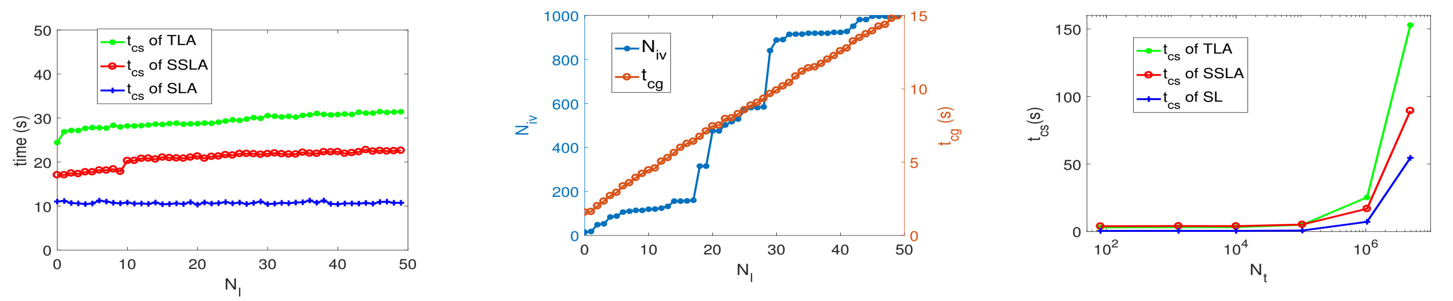


Figure 4: t_{cs} VS N_l in *min* case **Figure 5: N_{iv} and t_{cg} VS N_l in *full* case** **Figure 6: t_{cs} VS N_t in *full* case (log scale in X-axis)**

than 1 second). Although there are 14 views in total (the same as in the synthetic workload), only one covering set exists for most queries and the number of view mappings is far fewer than 14, leading to the short response time. This is the case when the views “partition" the relations.

Exp6. The last three rows of Table 11 shows that the number of covering sets N_{cs} and the time to generate them t_{cs} are still very small. However, the time for citation generation step t_{cg} is much larger than in Exp5, which is due to the fact that there is a join between large relations in the citation queries. Plus, for the same query, since more covering sets in the *full* case implies more citation queries, the non-negligible execution time for each citation query leads to the large gap between t_{cg} in *full* case and t_{cg} in *min* case. Thus Exp5 and Exp6 address Q4, i.e. the time performance and citation size of our approaches in the realistic scenarios.

Discussion. Although the performance and size of citations in the synthetic experiments is not acceptable for extreme values of view mappings (N_v), predicates (N_p) and lambda terms (N_l), Table 11 shows that for our realistic cases these values are all very small (less than 10). The number of tuples in the extended query result (N_t) is also very small (less than 10^4).

Revisiting the results of Section 5.2, Figures 2a and 2b show that, in the *full* case, when N_v is less than 10 the performance and size of citations is very reasonable for all three approaches: t_{cs} is less than

Table 11: Experimental results on realistic workloads

	Query	N_t	N_v	N_l	N_p	full case						min case					
						N_{cs}	N_{iv}	t_{cs} in TLA (s)	t_{cs} in SSLA (s)	t_{cs} in SLA (s)	t_{cg} (s)	N_{cs}	N_{iv}	t_{cs} in TLA (s)	t_{cs} in SSLA (s)	t_{cs} in SLA (s)	t_{cg} (s)
Exp5	Q0	8868	1	1	0	1	8868	0.25	0.18	0.15	0.58	1	3769	0.25	0.18	0.08	0.56
	Q1	1366	1	1	0	1	348	0.19	0.15	0.13	0.41	1	348	0.19	0.17	0.06	0.43
	Q2	2522	7	1	6	1	709	0.25	0.21	0.14	0.38	1	709	0.27	0.20	0.10	0.46
	Q3	120	8	2	6	1	74	0.18	0.16	0.13	0.38	1	74	0.20	0.16	0.07	0.47
	Q4	5748	7	1	6	7	2204	0.26	0.22	0.14	0.47	1	424	0.29	0.24	0.07	0.47
	Q5	1	8	2	6	1	1	0.16	0.14	0.12	0.37	1	1	0.18	0.16	0.06	0.45
	Q6	271	7	1	6	1	271	0.17	0.16	0.12	0.36	1	271	0.19	0.18	0.04	0.47
	Q7	521	1	1	0	1	521	0.16	0.14	0.13	0.41	1	521	0.15	0.13	0.05	0.41
Exp6	q1	4884	4	4	1	3	4917	1.19	1.18	1.15	11.31	1	33	1.19	1.22	0.67	1.46
	q2	27	4	4	1	3	6	1.81	1.72	1.69	10.40	1	2	1.73	1.71	1.18	1.41
	q3	7	2	3	0	2	14	0.94	0.91	0.90	2.31	1	7	0.95	0.92	0.45	1.37

30 seconds, the citation generation time t_{cg} is less than 5 seconds, and the covering set size N_{cs} is very small.

Figure 5 shows that when N_l is less than 10, t_{cg} is less than 5 seconds, which is also reasonable.

Figures 3a and 3b show that when N_p is less than 10, t_{cs} is less than 100 seconds for TLA and SSLA and holds steady at about 20 seconds for SLA. However, since N_t is usually less than 10^4 whereas the synthetic workloads shown used a setting of 10^6, t_{cs} for all three approaches in practice should be far less than 100 seconds. Furthermore, the citation generation time (t_{cg}) is less than 10 seconds when N_p is less than 10 even for the large result size.

6 CONCLUSIONS

In this paper, we build on the notion of citation views presented in [7] to give a semantics for citations to general queries based on *covering sets of mappings* between the views and the input query. We present *three approaches* to implementing citation views and describe *alternative policies* for the joint, alternate and aggregated use of citation views.

Extensive experiments were performed using *synthetic* as well as *realistic* citation views and queries for two different choices of policies. The experiments explore the tradeoffs between the approaches, and show that the choice of policy has a huge effect both on performance and on the size of the resulting citations. In particular, when the "best" citation is chosen rather than using "all possible" citations there is an order of magnitude speedup. The realistic cases show that all three approaches are feasible, although reasoning at the schema level results in a 2-3x performance gain at the expense of generating citations to individual tuples.

Practical experiences. A characteristic of the realistic settings (GtoPdb and DBLP-NSF) is that the view definitions are select-project queries over single relations. It is easily shown that if such select-project views *partition* the attributes of each relation so that no attribute appears in more than one view definition, there will be a *unique* covering set of mappings for each query. In this case it is straightforward to calculate the covering set, and the size of the citation will be driven by the 1) the number of relational subgoals in the query, 2) the number of partitions for each relation, and 3) the number of different lambda-terms in the matched views. Note that the number of partitions and the number of lambda-terms can be controlled by the DBA when constructing the citation views.

Since the size of the citation for general queries is an obvious concern, the DBA may also wish to specify "large subset" view queries in addition to partitioning view queries. For example, in GtoPdb the views $V1$ and $V2$ mirror what is currently done: each tuple in Family corresponds to a web-page as does each tuple in Family_Intro. However, when "large subsets" of the Family or Family_Intro relation are returned, a more succinct citation can be used, captured by the view $V3$.[8] If "large subset" view queries are specified for each relation in addition to the partitioning view queries and $+^R$ is *min*, calculating the covering set is still very efficient and the size of citations will be kept small (roughly the number of relational subgoals in the body of Q).

Although the cost function used in *min* can be difficult to understand (and therefore predict what the citation will be) for the synthetic case, it is straightforward when partitioning and "large subset" views are used in the practical case: For each relation in the body of the query, if a large subset of the relation is returned (where large is a parameter that can be set by the DBA) use the "large subset" view, otherwise use the parameterized view. This is analogous to the use of "*et al*" in traditional citations when the author list is longer than 3 people.

Future work. In future work, we would like to explore the connection of citation to *provenance*. Since both provenance and citations are annotations on tuples, it may be possible to reason over the provenance polynomials [10] of view definition tuples and those of result tuples to determine whether the result tuple carries the view tuple's citation annotation.

Versioning is also crucial for ensuring that cited data can be reconstructed. However, these techniques should be adapted for data citation, which requires versioning to be triggered when a user cites a data entry and only needs to record change on the cited data. Thus interesting optimizations may be possible in this context.

Finally, we would like to explore how citations can be integrated into *data science environments*, in which queries are interleaved with analysis steps. This is is difficult since provenance is not well understood in the context of machine learning algorithms.

[8] Views $V4$ and $V5$ are unrealistic, and were only introduced for pedagogical reasons.

REFERENCES

[1] *Out of Cite, Out of Mind: The Current State of Practice, Policy, and Technology for the Citation of Data*, volume 12. CODATA-ICSTI Task Group on Data Citation Standards and Practices, 2013.
[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
[3] A. Alawini, L. Chen, S. B. Davidson, N. Portilho, and G. Silvello. Automating data citation: the eagle-i experience. In *Proc. of the ACM/IEEE Joint Conference on Digital Libraries (JCDL 2017)*, pages 169–178, 2017.
[4] A. Alawini, S. B. Davidson, W. Hu, and Y. Wu. Automating data citation in citedb. *PVLDB*, 10(12):1881–1884, 2017.
[5] P. Buneman, S. B. Davidson, and J. Frew. Why data citation is a computational problem. *Communications of the ACM (CACM)*, 59(9):50–57, 2016.
[6] P. Buneman and G. Silvello. A Rule-Based Citation System for Structured and Evolving Datasets. *IEEE Data Eng. Bull.*, 33(3):33–41, 2010.
[7] S. B. Davidson, D. Deutsch, T. Milo, and G. Silvello. A model for fine-grained data citation. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Online Proceedings*, 2017.
[8] FORCE-11. *Data Citation Synthesis Group: Joint Declaration of Data Citation Principles.* FORCE11, San Diego, CA, USA, 2014.
[9] J. Goldstein and P. A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, pages 331–342. ACM Press, 2001.
[10] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance Semirings. In *Proc. of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 31–40, 2007.
[11] P. Groth, A. Gibson, and J. Velterop. The Anatomy of a Nanopublication. *Inf. Serv. Use*, 30(1-2):51–56, 2010.
[12] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
[13] J. Klump, R. Huber, and M. Diepenbroek. DOI for Geoscience Data – How Early Practices Shape Present Perceptions. *Earth Science Inform.*, pages 1–14, 2015.
[14] S. Pröll and A. Rauber. Scalable data citation in dynamic, large databases: Model and reference implementation. In *Proc. of the 2013 IEEE International Conference on Big Data*, pages 307–312, 2013.
[15] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 105–112, 1995.
[16] G. Silvello. A Methodology for Citing Linked Open Data Subsets. *D-Lib Magazine*, 21(1/2), 2015.
[17] G. Silvello. Learning to Cite Framework: How to Automatically Construct Citations for Hierarchical Data. *Journal of the American Society for Information Science and Technology (JASIST)*, 68(6):1505–1524, 2017.
[18] N. Simons. Implementing DOIs for Research Data. *D-Lib Magazine*, 18(5/6), 2012.
[19] P. Slavik. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–276, 1997.

A APPENDIX: APPROACHES

A.1 Details of Approaches: *full* case

A.1.1 Details of TLA. TLA consists of several steps: preprocessing, extended query execution, reasoning, population and aggregation.

When a user query Q is evaluated, the *preprocessing step* derives all possible view mappings and extends the schema of Q for checking the validity of view mappings and groupings. Details for deriving all possible view mappings are shown in Algorithm 1 and those for extending user queries are shown in Algorithm 2. These two algorithms form the preprocessing step.

After executing the extended query (*query execution step*), for each tuple $t \in Q_{ext1}(D)$, the valid view mappings are derived. The details are shown in Algorithm 3.

After calculating the valid view mapping sets for each relation B_i, the covering sets are derived by combining the view mappings to cover as many relations and distinguished variables of Q as possible; afterwards, duplicates are removed. The details are shown in Algorithm 4. Algorithms 3 and 4 form the *reasoning step.* After reasoning once for each group, the resulting covering sets are then propagated to all the tuples in the group (*population step*).

Algorithm 1: Deriving all possible view mappings

```
  Input  : a set of views: $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$, user query:
           $Q(\bar{X}) :- B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
  Output : the set of all possible view mapping $\mathcal{M}$
1 Initialize $\mathcal{M} = \{\}$
2 for each view $V \in \mathcal{V}$ do
3     Derive all possible view mappings from $V$ to $Q$ that follows definition 3.2
      and the second condition in definition 3.4 and add them to $\mathcal{M}$.
4 end
5 for each view mapping $M \in \mathcal{M}$ do
6     Derive lambda terms $L(M)$ and the predicates $condition(M)$ under $M$
7 end
8 return $\mathcal{M}$
```

Algorithm 2: Extend the schema of query in TLA

```
   Input  : The set of all the possible view mappings $\mathcal{M}$, user query:
            $Q(\bar{X}) :- B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
   Output : The extended query $Q_{ext1}(\bar{X}')$
1  Initialize the schema of the extended query $\bar{X}' = \bar{X}$
2  for each view mapping $M \in \mathcal{M}$ do
3      Add all lambda terms in $L(M)$ to $\bar{X}'$
4      Add boolean expressions of all the global conditions in $condition(M)$ to
       $\bar{X}''$
5  end
6  for each relation $B_i$ in the body of $Q$ do
7      Add the view vectors $Vec(B_i)$ to $\bar{X}'$.
8  end
9  Construct the extended query $Q_{ext1}$ with the following form:
10 $Q_{ext1}(\bar{X}') :- B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
11 return $Q_{ext1}(\bar{X}')$
```

Algorithm 3: Determine the valid view mappings in TLA

```
   Input  : Database instance D, the set of all the possible view mappings $\mathcal{M}$, the
            schema of the extended query $Q_{ext1}$: $(\bar{X}')$, and a tuple $t \in Q_{ext1}(D)$
   Output : A set of valid view mapping sets $\mathcal{M}(\mathcal{M}(t))$, a set of maximally
            covered relations $MCR(t)$
1  Initialize $\mathcal{M}(\mathcal{M}(t)) = \{\}$
   /* $\mathcal{M}(\mathcal{M}(t))$ denotes a set of view mapping sets */
2  for each view vector $Vec(B_i)(i = 1, 2, \ldots, m)$ do
3      Denote the set of all the valid view mappings from $Vec(B_i)$ as $\mathcal{M}_i(t)$
       Initialized $\mathcal{M}_i(t) = \{\}$
4      for each annotated view $V$ in $Vec(B_i)$ do
5          for each view mapping $M$ that the view $V$ is involved in do
6              check whether each view mapping $M$ satisfies:
                 (1) the first condition in the definition 3.4 by
                     checking whether all of the boolean expressions
                     of the global predicates in $condition(M)$ are true and
                 (2) $B_i$ is covered by the mapping $M$ and
                 (3) if $M$ covers more than one relations, $V$ should appear in
                     every view vector of those relations.
               if $M$ follows all the three rules above then
                   add $M$ to $\mathcal{M}_i(t)$
               end
7          end
8      end
9      if $\mathcal{M}_i(t)$ is not empty then
10         add $\mathcal{M}_i(t)$ to $\mathcal{M}(\mathcal{M}(t))$
11         add $B_i$ to $MCR(t)$
12     end
13 end
14 return $\mathcal{M}(\mathcal{M}(t))$ and $MCR(t)$
```

After the reasoning and the population steps, every tuple in $Q_{ext1}(D)$ is annotated with covering sets. The covering sets in the tuple level are then aggregated to derive the covering sets for the entire query (*aggregation step*, see Algorithm 5 for details). The aggregated covering sets are then converted to formatted citations using Algorithm 6. All the steps for TLA are shown in Algorithm 7.

Algorithm 4: Deriving the covering sets in TLA

```
Input  : Database instance D, The set of all the possible view mappings $\mathcal{M}$, the
         schema of $Q$: $\bar{X}$ , a tuple $t \in Q_{ext1}(D)$, a set of valid view mapping
         sets $\mathcal{M}(\mathcal{M}(t))$ and a set of maximally covered relations $MCR(t)$
Output : A set of covering sets $C(t)$
1  Initialize $C(t) = \{\}$
2  for each view mapping set $\mathcal{M}_i(t) \in \mathcal{M}(\mathcal{M}(t))$, each relation $R(\bar{X_R})$ from
     $MCR(t)$ do
3      for each variable $v$ from $\bar{X_R} \bigcap \bar{X}$ do
4          find a set of view mappings $\mathcal{M}_{\mathcal{R}} \in \mathcal{M}_i(t)$ in which each view
             mapping can cover $v$
5          if $\mathcal{M}_{\mathcal{R}}$ is not $\emptyset$ then
6              $C(t) = cross\_product(C(t), \mathcal{M}_{\mathcal{R}})$
7          end
8      end
9  end
10 for any two view mapping sets $C_i$ and $C_j$ in $C(t)$ do
11     if $C_i \subset C_j$ then
12         remove $C_i$ from $C(t)$
13     end
14 end
15 return $C(t)$
```

Algorithm 5: Deriving covering sets for the entire query

```
Input  : Group $G_i$, the covering sets in $G_i$: $C(G)$, The operator: $Agg$, current
         aggregated covering sets: $AC(Q)$
Output : Updated aggregated covering sets: $AC(Q)$
1  if $AC(Q) = \emptyset$ then
2      return C(G)
3  else
4      if $Agg$ is $intersection$ then
5          return $AC(Q) \bigcap C(G)$
6      else
7          return $AC(Q) \bigcup C(G)$
8      end
9  end
```

Algorithm 6: Details of citation generation step

```
Input  : a set of groups $\mathcal{G}$, the aggregated covering sets: $AC(Q)$, the extended
         query result $Q_{ext}(D)$
Output : A set of citations $C(Q)$
1  Initiate $C(Q) = \{\}$
2  for each group $G \in \mathcal{G}$ do
3      find all the covering sets $AC(G) \in AC(Q)$ so that each covering set in
         $AC(G)$ are valid for the group $G$
4      Get the values of lambda terms for views from covering sets $AC(G)$ by
         using all the tuples in $G$
5  end
6  for each covering set $AC \in AC(Q)$ do
7      Generate citation $C$ using the values of lambda terms and the citation
         queries corresponding to the each view in $AC$.
8      Put the citation $C$ in $C(Q)$
9  end
10 remove duplicates in $C(Q)$
11 return $C(Q)$
```

Algorithm 7: Details of TLA

```
Input  : a set of views: $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$, Database instance: $D$,
         Interpretation of aggregation: $Agg$, user query:
         $Q(\bar{X}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
1  Derive all the possible view mappings $\mathcal{M}$ using Algorithm 1
2  Derive $Q_{ext1}(\bar{X}')$ using Algorithm 2
3  Execute query $Q_{ext1}(\bar{X}')$ in $D$:
4  Tuples in $Q_{ext1}(D)$ are grouped during the execution of $Q_{ext1}(\bar{X}')$. Suppose
     there are $r$ groups, $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$
   /* Aggregated covering sets                                  */
5  $AC(Q) = \{\}$
6  for each $G$ in $\mathcal{G}$ do
7      select a representative tuple $t$ from $G$.
8      Determine the valid view mapping sets $\mathcal{M}(\mathcal{M}(t))$ and maximally covered
         relations $MCR(t)$ using Algorithm 3
9      Deriving all the covering sets $C(t)$ using Algorithm 4.
10     populate the resultant $C(t)$ to all the other tuples $t'$ in $G$.
11     Update $AC(Q)$ by using Algorithm 5
12 end
13 Generate formatted citations for covering sets $AC(Q)$ using Algorithm 6.
```

A.1.2 Details of SSLA.

A.1.2 Details of SSLA. The SSLA preprocessing step is almost the same as that of TLA, but the way in which the user query is extended is quite different. In order to check the validity of the view mappings for each tuple and group the tuples, the boolean expressions of both the local and global predicates of relevant views are included and no view vectors are available for extending the query.

The extended query instance $Q_{ext2}(D)$ is retrieved from the database by the *extended query execution step*. Valid view mappings are derived based on the extra columns in the schema of the extended query. Details are shown in Algorithm 8.

After deriving the set of valid view mappings, we obtain the covering sets using a similar strategy to the one used in TLA. Details are presented in Algorithm 9. Algorithms 8 and 9 form the *reasoning step* for SSLA.

Algorithm 8: Determine the valid view mappings in SSLA

```
Input  : Database instance D, The set of all the possible view
         mappings $\mathcal{M}$, the extended query: $Q_{ext2}(\bar{X}')$ :
         $-B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$, and a
         tuple $t \in Q_{ext2}(D)$
Output : A set of valid view mappings $\mathcal{M}(t)$, a set of maximally
         covered relations $MCR(t)$
1  Initialize $\mathcal{M}(t) = \{\}$
2  Initialize $MCR(t) = \{\}$
3  for each view mapping $M$ in $\mathcal{M}$ do
4      check whether $M$ satisfies the first condition in the definition 3.4
         by checking whether all of the boolean expressions of
         $condition(M)$ are true
5      if $M$ follows the rule above then
6          add $M$ to $\mathcal{M}(t)$
7      end
8      add all the relations in $Q$ that $M$ covers into $MCR(t)$
9  end
10 return $\mathcal{M}(t)$ and $MCR(t)$
```

The *population*, *aggregation* and *citation generation* steps in SSLA are the same as in TLA. All the steps for SSLA are shown in Algorithm 10.

Algorithm 9: Deriving the covering sets in SSLA

```
Input  : Database instance D, The set of all the possible view mappings $\mathcal{M}$, the
         schema of $Q$: $\bar{X}$, a tuple $t \in Q_{ext2}(D)$, a set of valid view mapping
         sets $\mathcal{M}(t)$
Output : A set of covering sets $C(t)$
1  Initialize $C(t) = \{\}$
2  for each relation $R(\bar{X_R})$ from $MCR(t)$ do
3      for each variable $v$ from $\bar{X_R} \bigcap \bar{X}$ do
4          find a set of view mappings $\mathcal{M_R} \in \mathcal{M}(t)$ in which each view
             mapping can cover $v$
5          if $\mathcal{M_R}$ is not $\emptyset$ then
6              $C(t) = cross\_product(C(t), \mathcal{M_R})$
7          end
8      end
9  end
10 for any two view mapping sets $C_i$ and $C_j$ in $C(t)$ do
11     if $C_i \subset C_j$ then
12         remove $C_i$ from $C(t)$
13     end
14 end
15 return $C(t)$
```

Algorithm 10: Details of SSLA

```
Input  : A set of views: $\mathcal{V} = \{V_1, V_2, ..., V_k\}$, Database instance: $D$,
         interpretation of aggregation: $Agg$, user query:
         $Q(\bar{X}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
1  Derive all the possible view mappings $\mathcal{M}$ using Algorithm 1
2  Derive extended query $Q_{ext2}(\bar{X}')$ using the algorithm similar to Algorithm 2
3  Execute query $Q_{ext2}(\bar{X}')$ in $D$:
4  Tuples in $Q_{ext2}(D)$ are grouped during the execution of $Q_{ext2}(\bar{X}')$. Suppose
     there are $r$ groups, $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$
   /* Aggregated covering sets */
5  $AC(Q) = \{\}$
6  for each group $G \in \mathcal{G}$ do
7      select a representative tuple $t$ from $G$
8      Determine the valid view mappings $\mathcal{M}(t)$ and the maximally covered
         relations $MCR(t)$ using Algorithm 8
9      Derive all the covering sets $C(t)$ using Algorithm 9
10     Populate the resultant $C(t)$ to other tuples $t'$ in $G$
11     Update $AC(Q)$ using Algorithm 5
12 end
13 Generate formatted citations for covering sets $AC(Q)$ using Algorithm 6.
```

A.1.3 Details of SLA. The major difference between SLA and the other two approaches is that SLA must remove view mappings with logically stricter predicates than the input query, and must also check foreign key constraints. For the remaining "valid" view mappings, the reasoning step is similar to the one described in Algorithm 9, so the details are ignored. The user query is then extended to include the lambda terms of valid view mappings (*query execution step*) so that view parameters can be evaluated, which is similar to Algorithm 2 and thus also ignored here.

A.2 Details of approaches: *min* case

As discussed in Section 4.4, finding the minimum-cost covering set is an NP-complete problem, which we approximate using a greedy algorithm [19] in the implementations. The details of how it is used in TLA and SSLA are slightly different from that of SLA.

A.2.1 Details of TLA/SSLA. For a tuple t in $Q_{ext1}(D)$ or $Q_{ext2}(D)$, after applying Algorithm 8, a set of valid view mappings will be derived. Then a well-known greedy algorithm for set cover is applied to find an approximate optimal solutions. The details are shown in Algorithm 11

Algorithm 11: Greedy algorithm using cost function

```
Input  : A set of valid view mapping $\mathcal{M}(t)$, a set of maximally covered relation
         $MCR(t)$.the view set $\mathcal{V}$
Output : A set of view mappings $C(t)$
   /* create a set to contain the selected view mappings, the result
      of which should be the covering set with minimal cost */
1  Initialize $C(t) = \{\}$
2  Derive a set of maximally covered distinguished variables $MCH(t)$
3  while $MCR(t) \neq \Phi$ and $MCH(t) \neq \Phi$ do
       /* use $Hv(M)$ and $R(M)$ to denote the distinguished
          variables and relations that view mapping $M$ covers in
          the query */
4      select $M$ from $\mathcal{M}(t)$ that can minimize
         $\frac{cost(M)}{|Hv(M) \bigcap MCH(t)|+|R(M) \bigcap MCR(t)|}$
5      $MCH(t) = MCH(t) \backslash Hv(M)$, $MCR(t) = MCR(t) \backslash R(M)$
6      add $M$ to $C(t)$
7  end
8  return $C(t)$
```

A.2.2 Details of SLA. Given a query Q, to speed up the process of finding its min-cost covering set we build a query lattice index ($\mathcal{L}_Q$) which is, in the worst-case scenario, the power set (without the empty set) of the relational subgoals of the query. Each node in this index is a pair $\langle key, value \rangle$ where the key is a set containing the subset of relational subgoals of Q and the $value$ is a subquery. Tops are the elements with no supersets and roots are the elements without subsets. We can further prune the lattice index by removing nodes which contain relations that cannot be covered by any views. Moreover, three lattice indexes partitioning the views (tables, distinguished variables and lambdas) are built off-line and used for fast-filtering the views in the reasoning step.

In Algorithm 12 we report the main steps of the greedy algorithm. In this algorithm we use two further functions. The first one is called `getSubquery`$(\mathcal{L}_Q, T_V)$, which takes the pruned query lattice index $\mathcal{L}_Q$ and a set of tables T_V as input and returns a subquery which uses only the given tables. `getSubquery` gets the *roots* of $\mathcal{L}_Q$, selects the nodes associated to queries which have at least one table in T_V and returns the query associated with their lowest common ancestor node. The other function is `GL01`, implementing the approach presented in [9], which checks if V is a valid cover for Q. This is achieved by comparing the relations, predicates, distinguished variables and foreign key constraints of V and Q.

B APPENDIX: DATASETS

B.1 DBLP-NSF dataset

Schema of relations. We show the parts of schema, which are touched by the views and user queries used in the experiments:

$dblp\_paper(pkey, ptitle, pyear, pconf)$
$dblp\_conference(ckey, cname, cdetail)$
$dblp\_author(aname, pkey)$
$grants\_awards(award\_id, award, effective\_date, expiration\_date, amount, instrument, org, officer, abstract, \ldots)$
$paper\_awards(award\_id, pconf, paper, author, pyear, pkey)$

Views. The views used in the experiments are defined below (not all of the attributes are listed to save space).

Algorithm 12: Greedy algorithm using cost function

```
Input  : the view set $\mathcal{V}$, the pruned query lattice $\mathcal{L}_Q$, the pruned query $Q(\bar{X})$
Output : A set of views $C$
$C \leftarrow \emptyset$;
$T_Q \leftarrow$ get table set of $Q$;
foreach $V_i \in \mathcal{V}$ do
    /* Assign the weight to the views */;
    $w_i \leftarrow |T_{V_i} \setminus T_Q| + 1$;
end
while $T_Q \neq \emptyset$ and $\mathcal{V} \neq \emptyset$ do
    foreach $V_i \in \mathcal{V}$ do
        /* Assign the cost to the view */;
        $c_i = \frac{w_i}{|\{C \cup \{T_{V_i} \cap T_Q\}\} \setminus C|}$;
    end
    Choose the set of views $\mathcal{V}_T$ sharing the minimum cost $c$;
    $C_{tmp} \leftarrow \emptyset$;
    foreach $V_i \in \mathcal{V}_T$ do
        $Q_t \leftarrow$ getSubquery($\mathcal{L}_Q$, $T_{V_i}$);
        if GL01($V_i$, $Q_t$) then
            $C_{tmp} \leftarrow C_{tmp} \cup V_i$;
        else
            $\mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
        end
    end
    if $|C_{tmp}| > 1$ then
        Choose the view $V_i \in C_{tmp}$ minimizing $|\bar{Y}_i \setminus \bar{X}| + |\lambda_{V_i} \triangle \lambda_Q|$;
        $C \leftarrow C \cup V_i, T_Q \leftarrow T_Q \setminus T_{V_i}, \mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
    else
        $C \leftarrow C \cup V_i, T_Q \leftarrow T_Q \setminus T_{V_i}, \mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
    end
end
return $C$
```

$$\begin{aligned}
&\lambda pk.V1(pk, pt, pconf) &&: -dblp\_paper(pk, pt, pyear, pconf)\\
&\lambda ck.V2(ck, cn, detail) &&: -dblp\_conference(ck, cn, detail)\\
&\lambda py, cn.V3(pk, pt, py, cn) &&: -dblp\_paper(pk, pt, py, pconf),\\
& &&\quad dblp\_conference(ck, cn, detail),\\
& &&\quad cn = pconf\\
&\lambda aid.V5(aid, award, abs, amt, eff\_date, ex\_date) && \\
& &&: -grants\_awards(aid, award, eff\_date,\\
& &&\quad ex\_date, amt, instr, org, officer, abs, \dots)\\
&\lambda eff\_date, org.V5(aid, award, abs, amt, eff\_date, ex\_date) && \\
& &&: -grants\_awards(aid, award, eff\_date,\\
& &&\quad ex\_date, amt, instr, org, officer, abs, \dots)\\
&\lambda pk.V6(pk, pt, pconf) &&: -dblp\_paper(pk, pt, pyear, pconf)
\end{aligned}$$

User queries. The three user queries used in the experiments are listed below (not all the attributes are listed due to space limit).

$$\begin{aligned}
&q1(pt) : -dblp\_paper(pk, pt, py, pconf), dblp\_conference(ck, cn, detail),\\
&\qquad cn = pconf, pconf = \text{`}VLDB\text{'}\\
&q2(pt) : -dblp\_paper(pk, pt, py, pconf), dblp\_conference(ck, cn, detail),\\
&\qquad dblp\_author(an, pk2), cn = pconf, pk = pk2, py = 2017, an = \text{`}X\text{'}\\
&q3(award) : -grants\_awards(aid, award, \dots),\\
&\qquad\quad paper\_awards(aid2, pconf, \dots), aid = aid2,\\
&\qquad\quad pconf = \text{`}VLDB\text{'}
\end{aligned}$$

B.2 GtoPdb dataset

Schema of relations. The schema of GtoPdb can be found on line, which is ignored here due to space limit.

User queries. The queries used in the experiments are:

$$\begin{aligned}
&Q1(oid1, ln) : -gpcr(oid1, \dots), interaction(oid2, lid2, \dots), ligand(lid1, ln, \dots)\\
&\qquad oid1 = oid2, lid1 = lid2\\
&Q2(oid1, ln) : -gpcr(oid1, \dots), interaction(oid2, lid2, \dots),\\
&\qquad ligand(lid1, ln, approv, \dots), lid1 = lid2, approved = \text{`}t\text{'}\\
&Q3(oid1, on) : -object(oid1, on, \dots), interaction(oid2, lid2, pri\_target, \dots),\\
&\qquad ligand(lid1, ln, \dots), oid1 = oid2, lid1 = lid2, pri\_target = \text{`}t\text{'}\\
&Q4(oid1, on) : -object(oid1, on, \dots), interaction(oid2, lid2, median, \dots),\\
&\qquad ligand(lid2, ltype, \dots), oid1 = oid2, lid1 = lid2,\\
&\qquad ltype = \text{`}Natural\ product\text{'}, median \geq 8.0\\
&Q5(ty, oid1) : -object(oid1, \dots), interaction(oid2, ty, \dots),\\
&\qquad oid1 = oid2, ty = \text{`}Agonist\text{'}\\
&Q6(oid1, on) : -ligand(lid1, ln, \dots), object(oid1, on, \dots),\\
&\qquad interaction(oid2, lid2, \dots), lid1 = lid2,\\
&\qquad oid1 = oid2, ln = \text{`}12R - HETE\text{'}\\
&Q7(oid, oname) : -object(oid, oname, in\_gtip, \dots), in\_gtip = \text{`}t\text{'}\\
&Q8(lid, lname) : -ligand(lid, lname, in\_gtip, \dots), in\_gtip = \text{`}t\text{'}
\end{aligned}$$

Views. The views used in the experiments are shown below:

$$\begin{aligned}
&\lambda F.V1(F, N) &&: -Family(F, N, Ty, \dots)\\
&\lambda F1.V2(F1, N, R2) &&: -Family(F1, N, Ty, \dots), receptor2family(OID2, F2)\\
& &&\quad further\_reading(OID1, R1), reference(R2, title, \dots)\\
& &&\quad F1 = F2, R1 = R2, OID1 = OID2\\
&\lambda F1.V3(F1, N, FN, SN) &&: -Family(F1, N, Ty, \dots), subcommittee(C1, F2, \dots),\\
& &&\quad contributor(C2, FN, SN, \dots), F1 = F2, C1 = C2\\
&\lambda F1.V4(F1, N, overview) &&: -Family(F1, N, Ty, \dots)\\
& &&\quad grac\_family\_text(F2, overview, \dots), F1 = F2\\
&\lambda Ty.V5(F, N) &&: -Family(F, N, Ty, \dots)\\
&\lambda F2.V6(F2, N, Tx, ant) &&: -Family(F1, N, Ty, \dots), introduction(F2, tx, ant, \dots)\\
& &&\quad F1 = F2\\
&\lambda OID1.V7(OID1, comments) &&: -receptor\_basic(OID2, comments),\\
& &&\quad object(OID1, \dots), OID1 = OID2\\
&\lambda OID1.V8(OID1, RID, comments) &&: -transduction(OID2, TID1, comments),\\
& &&\quad object(OID1, \dots), transduction\_refs(TID2, RID)\\
& &&\quad OID1 = OID2, TID1 = TID2\\
&\lambda OID1.V9(OID1, tissues, tech, SN, RID) &&: -species(S1, SN, \dots),\\
& &&\quad tissue\_distribution(TD1, OID2, S2, tissues, tech, \dots),\\
& &&\quad tissue\_distribution\_refs(TD2, RID)\\
& &&\quad object(OID1, \dots), OID1 = OID2, S1 = S2, TD1 = TD2\\
&\lambda OID1.V10(OID1, tissues, tech, SN, RID) &&: -species(S1, SN, \dots),\\
& &&\quad functional\_assay(FA1, OID2, S2, tissues, \dots),\\
& &&\quad functional\_assay\_refs(FA2, RID)\\
& &&\quad object(OID1, \dots), OID1 = OID2, S1 = S2, FA1 = FA2\\
&\lambda OID1.V11(OID1, tissues, trans, amin, GN, GLN, Gloc, SN, RID) && \\
& &&: -species(S1, SN, \dots), object(OID1, \dots),\\
& &&\quad structural\_info(SI1, OID2, S2, trans, amin, GN, GLN,\\
& &&\quad Gloc, \dots), structural\_info\_refs(SI2, RID)\\
& &&\quad OID1 = OID2, S1 = S2, SI1 = SI2\\
&\lambda OID1.V12(OID1, ac) &&: -receptor\_basic(OID2, comments, ac, \dots),\\
& &&\quad object(OID1, \dots), OID1 = OID2\\
&\lambda LID.V13(LID, N, appro, iup, comments) && \\
& &&: -ligand(LID, N, appro, iup, comments, \dots),\\
&\lambda OID.V14(OID, N) &&: -object(OID1, N, \dots)
\end{aligned}$$