Data Citation: Giving Credit Where Credit is Due*

Yinjun Wu
University of Pennsylvania
wuyinjun@seas.upenn.edu

Abdussalam Alawini
University of Pennsylvania
alawini@seas.upenn.edu

Susan B. Davidson
University of Pennsylvania
susan@seas.upenn.edu

Gianmaria Silvello
University of Padua
silvello@dei.unipd.it

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:
Yinjun Wu, Abdussalam Alawini, Susan B. Davidson, and Gianmaria Silvello. 2018. Data Citation: Giving Credit Where Credit is Due. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196910

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 give 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.

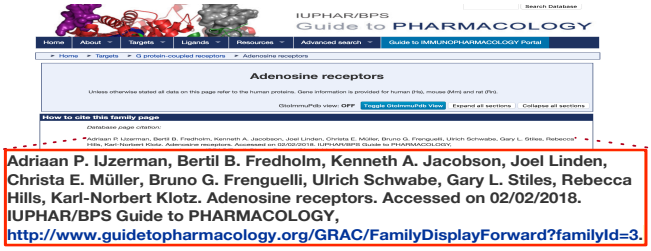


Figure 1: Sample data citation in GtoPdb

As an example, the IUPHAR/BPS Guide to Pharmacology [1] (GtoPdb) is a searchable database with information on drug targets and the prescription medicines and experimental drugs that act on them. The database content is organized by a hierarchy of families of drug targets; each family is curated by a (potentially different) group of experts. Information about a family is presented to users via a web-page view of the database, and a family-specific citation is presented at the bottom of the web-page (see Figure 1). The citation is generated from hard-coded SQL queries in the web-page form that retrieve the appropriate snippets of information from the database, which are then formatted to create a citation.

These snippets of information play several important roles. First, there is a *human* role: While the query and date of access (or some form of digital object identifier) are important for locating the query result, it does not give intuition about the content. For example, "Nature, 171,737-738" specifies how to locate an 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. Second, it enables *data bibliometrics*: Credit can be given to data creators and curators for the portion of the database to which they contributed, which encourages their continued contribution. This permits fine-grained citation counts extending the current practice of counting citations only at the dataset level (see the Data Citation Index by Clarivate Analytics [15]). Third, the snippets can capture *provenance* by including information about contributors/curators and other relevant information.

*Produces the permission block, and copyright information

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196910

[1] http://www.guidetopharmacology.org/

A number of scientific databases therefore specify (in English) what snippets of information to include in citations to certain web-page views of the data. Examples include the Reactome Pathway database [2] and eagle-i [3]. However, they do not automatically generate the citations, leaving it to the user to construct them by hand.

More advanced databases are wrestling with the issue of *automatically* generating citations to general queries over their data. For example, while GtoPdb generates citations for web-page views of the database (*frequent queries*) it does not do so for other *general queries*, although the developers have said they would like to enable this in the future [9, 12]. Other examples include Disgenet [4] and WikiPathways [5], both of which allow users to extract arbitrary subsets of data and wish to enable citation.

Goals and Challenges. The goal of our work is to develop a framework to *automatically generate citations to general queries*. This is challenging because there are many potential queries over a database, each accessing and generating different subsets of data. Each of these subsets may be attributable to different sets of people, and have different descriptive information. It is therefore infeasible to specify a citation to every possible query result. The idea that we explore in this paper is to specify citations for a small set of *frequent* queries (e.g. web page views), and use these to automatically construct citations for data returned by *general* queries.

Another goal is to *efficiently manage fine-grained citations*. Discussions with users show that they frequently want to refer to a subset of the query result rather than the entire result. For instance, in the neuro-imaging community it is quite common to query a system to get a set of relevant images and then manually narrow down the result set [22] (see also [27]). We therefore manage citations at the level of individual tuples in the query result, and lift the citation up to the level of any selected subset of the query result.

Approach. Our framework for data citation is based on *conjunctive queries* [3]. Conjunctive queries form the basis of languages associated with several different data models, enabling the framework to be used across a variety of different database system (including relational, XML, and RDF). The framework builds on the idea of *citation views* proposed in [12]. A citation view specifies what snippets of information to include and how to construct the citation for a particular query – or *view* – of the database. The intuition is that if a tuple in the result of a general query is "visible" in such a view, then the result tuple carries the view tuple's citation.

The architecture of our framework is shown in Figure 2. The DBA specifies citations for a small set of frequent queries (*Citation Views*). When a general query Q is submitted, the views are mapped to each tuple t in the result of Q. Sets of mapped views are then constructed that "cover" t (a *Covering Set*). The citations associated with each view in a covering set are then *jointly* used to construct a citation to t. Since there may be more than one covering set for t, our system reasons over *alternate* covering sets. Citations to individual tuples are then *aggregated* to form a citation to the selected query result. The joint (*), alternate ($+^R$), and aggregated (*Agg*) use of citation views are examples of *Policies* that are given by the DBA. We note (but do not discuss further) that the query

[2]http://www.reactome.org/pages/documentation/citing-reactome-publications/
[3]https://www.eagle-i.net/get-involved/for-researchers/citing-an-eagle-i-resource/
[4]http://www.disgenet.org/web/DisGeNET/menu
[5]https://www.wikipathways.org/index.php/Portal:Semantic_Web

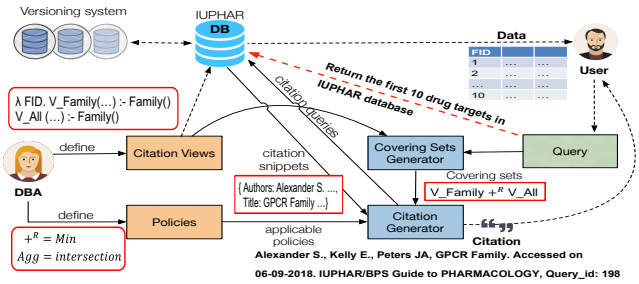


Figure 2: Data Citation Framework and Example

result must be available when the citation is later dereferenced, e.g. the database could be *versioned* and the query and version number included in the snippets of information in the citation.

We now motivate the alternate ($+^R$) and aggregated (*Agg*) use of citations, and will expand on this later in Section 3.

Example. Recall that GtoPdb organizes drug targets into families. Referring again to Figure 2, the DBA specifies two citation views (see red box above DBA), one which returns the citation to a single family (V_{Family}) and includes the contributors to the family, and the other which returns the citation to all families (V_{All}) and includes the curation committee of the entire project. Note that V_{Family} is *parameterized* [28] by the id of the family, indicated by the λ-term, meaning that each family has a different citation. In contrast, in V_{All} each family has the same citation. When a user asks a query over GtoPdb which returns a set of drug targets that spans multiple families, the citation system determines that both views could be used as covering sets for each tuple. After aggregating over the result set (specified as *intersection* over the view name, see red box below DBA), the covering set V_{Family} will generate a citation for the query result which includes all the contributors of each family in the result. In contrast, the aggregated citation using V_{All} would be a single citation shared by all families which includes the committee member. Both of these citations are returned by applying the policy *union* for $+^R$, and contributors to some family in the query result as well as the curation committee would be credited. In contrast, if the policy *min* were used for $+^R$ only the curation committee would appear. Note that the query as well as the date are included in the citation at the bottom right of the figure to enable the data to be retrieved at a later date.

This example illustrates why alternate covering sets might be desirable – to balance between the *size* and *specificity* of the final citation. It is also possible to ensure that citations are *unique* at design time by guaranteeing that there is at most one covering set for any input query (*partitioning views*, see Section 3.2).

Implementation. Since we implement fine-grained citations, each tuple in the query result may have a different citation (as illustrated by V_{Family} in the example above). This leads to two concerns: 1) *time* overhead, since the citation system is an interactive tool; 2) *size* of the citation.[6] To test whether fine-grained citations are feasible, we present three approaches to implementing citation views, in which the reasoning progressively shifts from the

[6]If citations are thought of as searchable digital objects rather than consuming space on paper, size may be less of a concern.

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. Based on these results, we conclude that generating citations for realistic citation views, queries and policies is effective both in terms of time overhead 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 framework for citation based on *conjunctive queries* [3] that can be used across many different types of databases, including relational, XML, and RDF (Section 3.2).
(2) A semantics for citations to general queries using *Citation Views* based on covering sets of mappings between the views and the input query (Section 3.3).
(3) Three approaches to implementing the *Covering Sets Generator* in Figure 2, which are then used to generate citations for general queries (Sections 4.1- 4.3). Two of the approaches enable fine-grained citations, while the last generates citations to the *entire* result.
(4) 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).
(5) Extensive experiments performed in the context of a relational database implementation 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 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.

The rest of the paper is organized as follows: Section 2 discusses related work in the digital libraries and database communities. The model and running example (GtoPdb) are presented in Section 3, along with a discussion of the relationship of our model to query rewriting using views. Section 4 describes the three approaches, and discusses different policies for joint, alternate, and aggregated use of citations. Section 5 presents experimental results. We conclude in Section 6. Appendix A and B contain details of implementations and the datasets used in the experiments respectively.

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 [16]. 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 the cited data, persistent identifiers and their 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 it describes. These were also included in a series of 14 recommendations by the Research Data Alliance (RDA) [29].

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) [23, 33]. While persistent identifiers enable the data to be located and, provided the cited data are somehow versioned, have associated guarantees of persistence (fixity), they do not constitute a full-fledged solution for data citation. Relevant examples are the Dataverse network and the DataCite initiative [2, 8]. They mint and assign DOI to datasets, but they do not handle dataset versioning, automatically generate snippets of information that are useful for human understanding (completeness), or address the issue of the variable granularity of data to be cited (e.g., subsets or aggregations). These and other deficiencies were noted in [9], which posed data citation as a *computational problem*.

Several proposals target **XML** data. The first is a *rule-based citation system* that exploits the hierarchical structure of XML to provide citations to XML elements [10]. The second uses *database views* to define citable units as a key to specifying and generating citations to XML elements [9]. This approach was then extended in [5] 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 [32].

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 [19]. 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 [31]. 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 [5].

Two approaches deal with citation for **relational databases**. In Pröll et al [26, 27], a query against the database returns a result set as well as an associated stable identifier which 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. This solution has been implemented for CSV as well as for relational databases. In particular, Pröll et al's approach addresses *identification*, *persistence* and *fixity* of data citation, but not *completeness* (recommendation 10 of RDA), which is our target. These ideas could be integrated with our approach by including stable identifiers [26, 27] (or DOIs[2, 8]) in the snippets of information in the citation.

The second approach [9] proposes a hierarchy of citable units that can be attached to parts of the database, and used to generate citations for user queries. This idea was formalized in [12], and an architecture and partial implementation were proposed in [6]. In this paper, we are extending on our previous work by presenting a semantics of *covering sets of mappings* using conjunctive queries, 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* [3]. Conjunctive queries are at the core of many query languages for relational, semi-structured, and graph-based data [7, 13], and therefore the framework extends well beyond relational systems; in particular, conjunctive queries were used in [5] to specify citations for the eagle-i RDF dataset. Conjunctive queries also simplify the reasoning used to generate citations for general queries.

We start by describing the GtoPdb database [21], which will be used as a running example throughout this section. We then discuss how *citation views* are specified for *frequent* queries and show how they can be used to generate citations for *general* queries, i.e. queries for which citations have not been specified. We also discuss a simple but common special case of *partitioning views* which avoids the problem of alternative citations to general query results. We conclude by discussing the connection between citation reasoning and query rewriting using views.

3.1 Running Example: GtoPdb

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 authored by a set of contributors; 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 contributors for the family. The citations for a family landing page and detailed introduction page may therefore differ.

The citation for GtoPdb as a whole is a traditional paper written by the database owners [21], a citation to a family page includes the contributors for the content of 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(FID, FName, Type)
FamilyIntro(FID, Text)
Person(PID, PName, Affiliation)
FC(FID, PID), FID references Family, PID references Person
FIC (FID, PID), FID references FamilyIntro, PID references Person
MetaData(Type, Value)

Intuitively, FC captures the contributors for 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').

3.2 Citation views

The citation framework is based on a set of citation views, which specify how citations are constructed for common queries against the database. A citation view specifies: 1) the data being cited (*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 uses the output of the citation queries to construct the citation in some appropriate format (e.g. BibTex, RIS or XML). The citation can be thought of as an *annotation* on every tuple in the view result.

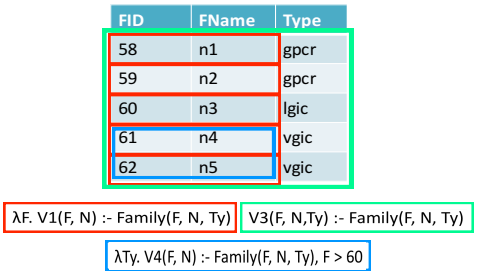


Figure 3: Effect of Parameters on Views

To simplify the reasoning for generating citations for general queries, the view definition is a (non-recursive) conjunctive query. The remaining components – the citation query and citation function – may be in any language, although throughout this presentation we illustrate citation queries using conjunctive queries.

The view definition and citation queries are optionally *parameterized*, where the parameters (*lambda variables*) appear as variables somewhere in the body of the query. [7] 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:

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

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 3 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 single view.

We assume that all queries (including view definitions) use fresh variables in every position; any *local* predicates on variables (i.e. those involving a single variable) and *global* predicates (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 contributors for the content of a family page while FIC captures the contributors who author the family introduction page:

$$\lambda F.\ C_{V1}(F,N,Pn) \quad :- Family(F,N,Ty), FC(F,C),$$

[7] Also called *binding patterns* in [28].

$Person(C, Pn, A)$

$\lambda F.\ C_{V2}(F, N, Tx, Pn)$ $:-\ Family(F, N, Ty), FamilyIntro(F, Tx), FIC(F, C), Person(C, Pn, A)$

$C_{V3}(X1, X2)$ $:-\ MetaData(T1, X1), T1 = \text{'}Owner\text{'}, MetaData(T2, X2), T2 = \text{'}URL\text{'}$

$\lambda Ty.\ C_{V4}(Ty, N, Pn)$ $:-\ Family(F, N, Ty), FC(F, C), Person(C, Pn, A)$

$\lambda Ty.\ C_{V5}(N, Ty, Tx, Pn)$ $:-\ Family(F, N, Ty), FamilyIntro(F, Tx), FIC(F, C), Person(C, Pn, A)$

The view to which a citation query is associated is given as a subscript, e.g. C_{V1} is associated with $V1$. To ensure 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 queries 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', contributors: ['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.

Partitioning Views. The sample views $V1$-$V5$ are more complex than that we have seen in practice, and are introduced for pedagogic reasons. The views currently used in GtoPdb are essentially $V1$ and $V2$, extended to include as head variables all attributes in Family and FamilyIntro, respectively. $\{V1, V2\}$ illustrates a simple but common case of a set of views that *partition* the database schema: Each attribute of each relation appears in at most one view. In contrast, $\{V1, V3, V4\}$ is not partitioning since the FName attribute of Family appears in all three views. As we will see in the next subsection, attributes which appear in multiple views lead to *alternative* citations for the query result, which may (or may not) be undesirable from the perspective of the DBA. In the case that views are select-project views of a single relation (e.g. $V1$-$V4$ above), it is easy to check whether they are partitioning (proof omitted). The DBA could therefore be warned if a given set of views for a relation would lead to alternative citations, and decide what they want.

3.3 General queries

To give a semantics to citations for *general queries*, we use the following intuition: *If a view tuple is visible in the query result, then the result tuple carries the view tuple's citation annotation.* To do this, we find maximal, non-redundant sets of valid mappings from the views to the input query (*covering sets*). For each such set of mappings, the citation is constructed by jointly using the citations of the views in the mappings. We formalize this as follows.

Definition 3.1. **View Mapping** Given a view definition V and query Q

$$\mathtt{V}(\bar{Y}) : -\mathtt{A}_1(\bar{Y_1}), \mathtt{A}_2(\bar{Y_2}), \ldots, \mathtt{A}_k(\bar{Y_k}), \mathtt{condition(V)}$$
$$\mathtt{Q}(\bar{X}) : -\mathtt{B}_1(\bar{X_1}), \mathtt{B}_2(\bar{X_2}), \ldots, \mathtt{B}_m(\bar{X_m}), \mathtt{condition(Q)}$$

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 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 $\phi(y) = x$ for some y.

Example 3.2. Consider the following query which finds the names of all 'gpcr' families that have an introduction page:

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

There are obvious view mappings from each of the views presented above to the body of Q. For example, one possible mapping, $M1$, maps the first (and only) subgoal of $V1$ to the first subgoal of Q and induces the mapping of variables $\phi(F) = F1,\ \phi(N) = N,\ \phi(Ty) = Ty$. Another mapping, $M4$, also maps the first subgoal of $V4$ to the first subgoal of Q and induces the mapping of variables $\phi(F) = F1,\ \phi(N) = N,\ \phi(Ty) = Ty$.

A view mapping will only be valid for a tuple in the query result if the relevant portions of the tuple matches the *local and global predicates* of the view and is *visible* in the view. To determine this, we must reason over *all* variables appearing in the body of the query as well as the view, and therefore introduce the *projection-free* notion of a query extension:

Definition 3.3. **Query Extension** Given a query

$$\mathtt{Q}(\bar{X}) : -\mathtt{B}_1(\bar{X_1}), \mathtt{B}_2(\bar{X_2}), \ldots, \mathtt{B}_m(\bar{X_m}), \mathtt{condition(Q)}$$

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

$$\mathtt{Q_{ext}}(\bar{X}') : -\mathtt{B}_1(\bar{X_1}), \mathtt{B}_2(\bar{X_2}), \ldots, \mathtt{B}_m(\bar{X_m}), \mathtt{condition(Q)}$$

where $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$. Note that $\bar{X} \subseteq \bar{X}'$.

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

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 a distinguished variable
- All lambda variables in V are mapped to variables in $\bar{X}'$.

Example 3.5. Suppose the tuple t=(58, 'n1', 'gpcr', 58, 'tx1') appeared in the result of Q_{ext} from Example 3.2. Then a view mapping for $V2$ would not be valid since (58, 'tx1') is not visible in the result ('n1'), and a view mapping for $V4$ would not be valid since the local predicate $F > 60$ is not met. However, the view mappings for $V1$ ($M1$), $V3$ ($M3$) and $V5$ ($M5$) would be valid.

Given a set of views $\mathcal{V}$, a query Q and a database instance D, a set of valid view mappings $\mathcal{M}(t)$ is built for each tuple $t \in Q(D)$ according to Definitions 3.1 and 3.4. Different view mappings from $\mathcal{M}(t)$ are then combined to create a *covering set* of views for t.

Definition 3.6. **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, $\{C_1, ..., C_k\}$.

Example 3.7. Returning to Example 3.5, the covering sets for t are $C_1 = \{M1\}$, $C_2 = \{M3\}$ and $C_3 = \{M5\}$. C_1 is parameterized by FID and would therefore generate different citations for each result tuple in Q_{ext}. C_2 is not parameterized and would therefore generate the same citation for each result tuple. C_3 is parameterized by Ty which matches a local predicate of Q_{ext} and would also generate the same citation for each result tuple. Note that C_3 is a "tighter" match than C_2 since there are tuples in $V3$ that do not appear in the query result whereas all tuples in $V5$ do appear.

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

Partitioning Views, revisited. The running example illustrates that there may be several alternative citations that can be associated with each tuple in the query result. However, if the views are partitioning, then there is a *unique* covering set for each tuple in the query result which meets the local predicates in matched views.

Example 3.8. Consider the following query and views:

$Q'(F1, N, Ty, Tx) :- Family(F1, N, Ty), FamilyIntro(F2, Tx), F1 = F2$
$\lambda F.V1'(F, N, Ty) :- Family(F, N, Ty)$
$\lambda F.V2'(F, Tx) :- FamilyIntro(F, Tx)$

$\{V1', V2'\}$ is partitioning, and is carried by each tuple t in the query result. However, since the views are parameterized by FID, the citations would potentially be different for each tuple in the query result, leading to a large citation result for the entire query.

The example above illustrates why, when general queries are allowed, the DBA may want to include additional, redundant views. For example, adding the following views:

$V3'(F, N, Ty) :- Family(F, N, Ty)$
$V4'(F, Tx) :- FamilyIntro(F, Tx)$

would lead to a choice of four covering sets. However, the DBA could give an interpretation of $+^R$ (a *policy*) which gave preference to the citation associated with $\{V3', V4'\}$ for Q' (since it leads to a single citation shared by all result tuples), but gave preference to the citation associated with $\{V1', V2'\}$ for a query which specified the FID (since the query result would contain at most one tuple, and the citation would be "precise" for that tuple). This is analogous to the use of "et al" in traditional citations when author lists are very long, when conciseness is preferred over specificity.

Finalizing the citation. 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 then 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

3.4 Query Rewriting Using Views: Discussion

Query rewriting using views has been used in many data management problems, in particular query optimization and data integration [20]. We now discuss the relationship between covering sets 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, e.g., minimizing the number of views.

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 view mappings 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 view mappings 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. Due to the close connection of citation reasoning to query rewriting using views, and recent work on this problem for both XML [11, 25] and RDF [14, 24], we believe our framework can be used in those models.

The most significant difference between citation reasoning and query rewriting using views is that citation reasoning supports tuple-level reasoning. The approaches described in the next section are therefore potentially applicable in any scenario where fine-grained reasoning is needed, e.g. fine-grained access control [30].

4 APPROACH

We now describe three approaches to implementing the citation model for general queries discussed in Section 3.3: *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

In order to generate citations, we first need to calculate the covering sets for each tuple in the query result. Covering sets are created from valid view mappings (Definition 3.4). The last two conditions in this definition can be easily checked using the view mapping M by comparing the schema of the view V and query Q. However, the first condition is harder since it must be checked tuple by tuple in the query result. Thus the satisfiability of (local and global)

predicates of view V under view mapping M will become the main concern in our approaches.

To facilitate checking local predicates, in the *tuple-level approach* the database schema is modified: A *view vector* column is added to each relation identifying all views in which a tuple potentially participates. For each view $V :- B_V$, V is added to the view vector of each tuple t in relation $R \in B_V$ whenever t satisfies the *local predicates* for V. This reduces the overhead for checking the *local predicates* at query time, and filters out invalid view mappings early. Any *global predicates* are checked at query time.

Preprocessing step. When a query $Q :- B_Q$ is submitted, we first calculate all *possible* view mappings using the view and query schemas. Some of these mappings may become invalid for individual result tuples depending on whether global predicates 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 (which are used to evaluate parameterized views); 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 (which are used to filter out invalid view mappings based on global predicates).

Query execution step. The extended query, Q_{ext1}, is then executed over the database instance D, 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 3.2, 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 calculation of covering sets starts from considering all view mapping combinations and ends up with maximal and non-redundant sets, e.g. {M3, M2} and {M5} for the first tuple in $Q1_{ext1}(D)$ (other combinations like $\{M1, M2, M5\}$ is redundant and thus thrown away). The final query result with covering sets is shown in Table 6. View mappings of parameterized views are instantiated by passing the parameter values (e.g. M2(58) indicates view mapping of V2 for family_id=58). 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 predicates, deriving covering sets can be done 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 mappings.

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 extended to include 1) lambda variables under all the possible view mappings; and 2) columns representing the truth value of every global and local predicate. 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 set of valid view mappings for each tuple $t \in Q_{ext2}(D)$, $VM(t)$, is derived based on the truth values of the *global* and *local predicates* (all must be true for a view mapping to be in $VM(t)$). In the second phase, covering sets for t are calculated using the view mappings from $VM(t)$.

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

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)

appears in the body of $Q1$, so the only additional information is the local predicate test for V4 under mapping M4, $FID1 > 60$. The result of this test shows that M4 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 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.

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

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 set of covering sets that is valid for all possible instances of the database. The SLA implementation borrows ideas from the query rewriting using views techniques proposed in [17].

Reasoning step. The first phase of reasoning in SLA calculates the valid view mappings. 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.[8] 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 view mappings.

Example 4.3. Returning to Example 4.1, using schema level reasoning M4 would not be considered since the predicate $F > 60$ does not appear in Q and is therefore more restrictive. Note that the predicate 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) global predicates in M5 is also in Q. Hence the resulting covering set would be M3*M2 $+^R$M5. Now suppose the query was:

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

The local predicate of M4, $FID > 60$, is less restrictive than the corresponding local predicate of $Q2$, $FID > 70$. Hence M4 is a valid

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

view mapping for $Q2$. However there is now no valid mapping from V5 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 lambda variables that do not appear as distinguished variables under all the valid view mappings. The extended query, Q_{ext3}, is then evaluated to construct covering sets and thus 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 predicates and SSLA extends with both global and local predicates. 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 only wish to cite the entire query result; it also may not generate the "most specific" citation if all tuples in the query result happens to satisfy the local and global 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 abstract operators $*$, $+^R$, $+$ and *Agg* can have different interpretations. For example, $*$ can be *join* or *union*; $+^R$ can be *union* or *min*; $+$ can be *union* and *Agg* can be *intersection* or *union*. 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, and thus reducing the overhead 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 $O(log\ n)$-approximate solution [34]. (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.

Table 8: Citations for sample view mappings

View	Result of citation function
M2(58)	{ID: '58', author: ['Mark'], Contributors: ['Poyner']}
M3	{author: ['Roger'], Contributors: ['Justo']}

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')}.[9]

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 is to generate covering sets for the query result (or for subsets of the query result). When *union* is used, the covering sets are calculated using the view mappings valid for some tuples; when *intersection* is used, the view mappings valid for the entire query result are involved in the construction of covering sets. 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, the citation {M5} would be applied over all types in the query instance, i.e. 'gpcr', 'vgic' and '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).

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 (See Table 8):
{ID: '58', author: ['Mark', 'Roger'], Contributors: ['Poyner', 'Justo']}.
If the interpretation is *Union*, the citation will be the set of objects:
{{ID: '58', author: ['Mark'], Contributors: ['Poyner']}, {author: ['Roger'], Contributors: ['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

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

a linux server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz and 64GB of central memory. Our code is publicly available [10] and uses pieces of code developed by the authors of [4].

Datasets. Our experiments used two datasets. The first is the GtoPdb database.[11] 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 DBLP—to their NSF funding grants—extracted from the National Science Foundation grant dataset. We refer to this database as DBLP-NSF, and have also made it publicly available.[12] The idea was to add funding information to traditional paper citations, and to be able to vary between citing the conference proceedings (if large number of papers from the same conference were in the result set) and citing individual papers. DBLP-NSF consists of 17 relations (authors, papers, 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*. 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 relational subgoals; 2) the number of tuples in the extended query result (N_t). We also implemented a *view generator* which takes as input: 1) the number of views (N_v); 2) the number of lambda terms in total (N_l); and 3) the total number of predicates (N_p). 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 from each view to the query.

For realistic workloads, we used citation views and general queries (Q0-Q7) from anticipated workloads of GtoPdb and DBLP-NSF. For GtoPdb, general queries were designed by consulting with the database owners, and views were designed based on its webpage 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 a single paper, single conference and single grant. General user queries (q1-q3) 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, the citation generation step produces a formatted citation, the time for which is denoted by t_{cg}. To evaluate the citation size, we measure the number of covering sets (N_{cs}) for the entire query. Table 9 provides a summary of this notation.

Table 9: Notation used in the experiments

Notation	Meaning
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_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

Our evaluation addresses the following 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 does N_t influence the time performance?
- EQ4: What is the performance and citation size of the three approaches in realistic scenarios?

5.2 Synthetic workloads

We first report on experimental results under synthetic workloads.

Exp1. The first experiment evaluates the effect of N_v on time performance and citation size. We configured the query generator to generate queries which produce a result set of about one million tuples ($N_t = 10^6$) using a subset of the product of four randomly selected relations. The view generator varied the number of views and ensured valid view mappings from each view to the query. Here, we do not consider other view features such as lambda terms and predicates.

Results. For the *full* case, thousands of covering sets are generated as the number of view mappings exceeds 30 (Figure 4a). As expected, the corresponding time to generate them (t_{cs}) increases exponentially. As shown in Figure 4b, the major overhead in t_{cs} is the reasoning time t_{re} when N_v exceeds 25, leading to a convergence of the three approaches.

Figure 4c shows the results for the *min* case, which has a huge speed-up compared to the *full* case. Notice that even with a large N_v, t_{cs} is acceptable (about 25 seconds).

[10] Our code is available at https://github.com/thuwuyinjun/Data_citation_demo.
[11] GtoPdb is available at http://www.guidetopharmacology.org/download.jsp.
[12] The DBLP-NSF dataset is available at https://data.mendeley.com/datasets/ycnngyv5bd.

Table 10: Experimental results on real workloads (full case)

Query	N_t	N_v	N_p	N_{cs}	t_{cs} in TLA (s)	t_{cs} in SSLA (s)	t_{cs} in SLA (s)	t_{cg} (s)
Q0	8868	1	0	1	0.25	0.18	0.15	0.58
Q1	1366	1	0	1	0.19	0.15	0.13	0.41
Q2	2522	7	6	1	0.25	0.21	0.14	0.38
Q3	120	8	6	1	0.18	0.16	0.13	0.38
Q4	5748	7	6	7	0.26	0.22	0.14	0.47
Q5	1	8	6	1	0.16	0.14	0.12	0.37
Q6	271	7	6	1	0.17	0.16	0.12	0.36
Q7	521	1	0	1	0.16	0.14	0.13	0.41
q1	4884	4	1	3	1.19	1.18	1.15	11.31
q2	27	4	1	3	1.81	1.72	1.69	10.40
q3	7	2	0	2	0.94	0.91	0.90	2.31

These 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. Since each covering set represents a possible citation, this also generates 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, we need to execute extra steps in the population step to store covering sets for each tuple.

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 as 15 and varied the total number of local predicates (and thus N_p) from 0 to 50.

Results. The number of predicates (N_p) influences the time performance of TLA and SSLA in two 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 create more groups in the extended query result, incurring more reasoning time (t_{re}). 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 5 and 6 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. In Figure 5, t_{qe} is included for TLA and SSLA, and shows that in the *min* case, increasing N_p results in slight increases in the (extended) query execution time for SSLA; thus t_{cs} in SSLA is only about twice that in TLA when N_p is up to 50. The slightly worse performance of SSLA 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 not necessary in TLA. The same is true for the *full* case (Figure 6).

For the *full* case, the reasoning time t_{re} becomes a major overhead as N_p increases for both TLA and SSLA. This is because more predicates can create more groups in the query result, which incurs more t_{re} in total. Unlike TLA and SSLA, SLA is not influenced by the predicates since the reasoning is at the schema level.

In terms of citation size, in the *full* case thousands of covering sets are generated when N_p is large since *Agg* is union, which drives the increase of the citation generation time t_{cg}. The trend is almost the same as Figure 4a (except for the label of x-axis) and thus the figure is omitted. Thus EQ2 is partially answered, i.e. what the effect of N_p is on the time performance and citation size.

Exp3. This experiment evaluates the scalability of our approaches in terms of time performance by varying N_t. The view generator randomly generates 15 views ($N_v = 15$) 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.

Results. Figure 7 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 SLA is less than 60 seconds, addressing EQ3.

Discussion. These experimental results address EQ1, EQ2 and EQ3.[13] 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 all three approaches are very slow in the reasoning step since all possible citations are generated.

5.3 Realistic workloads

We now report experiments performed on the realistic datasets. Table 10 shows all experimental results for the *full* case. Results for the *min* case are only marginally better, and are not shown.

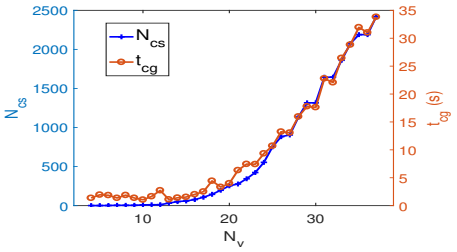
Exp4. This experiment evaluates how well the proposed approaches handle realistic workloads in the GtoPdb dataset. In this experiment, 14 views were created and each view has one associated citation query according to the web-page views. Eight user queries (Q0-Q7) were collected from the owners of GtoPdb.

Results. The first eight rows of Table 10 shows the result of Exp4. The time to generate covering sets t_{cs} and the time for the citation generation step t_{cg} for all the queries are very small (less than 1 second). Although there are 14 views in total, 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.

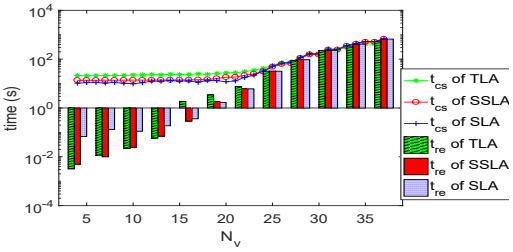
Exp5. This experiment is conducted on the DBLP-NSF dataset. Six views are used, each of which is associated with 1-2 citation queries. Three typical user queries are used as input. The first ($q1$) asks for the titles of papers in a certain conference (e.g. VLDB), while the second (q2) retrieves the 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).

Results. The last three rows of Table 10 show 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 that in Exp4, which is due to the fact that there is a join between large relations in some of the citation queries. Thus Exp4 and Exp5 address EQ4, i.e. the time performance and citation size of our approaches in the realistic scenarios.

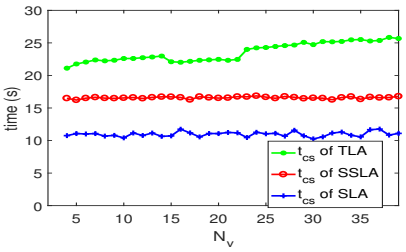
[13] We also ran experiments to determine the effect of the number of lambda terms, but found that it does not have significant influence.



(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 4: time performance and citation size VS number of view mappings

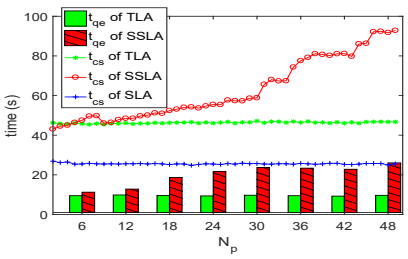


Figure 5: t_{cs} and t_{qe} VS N_p in *min* case

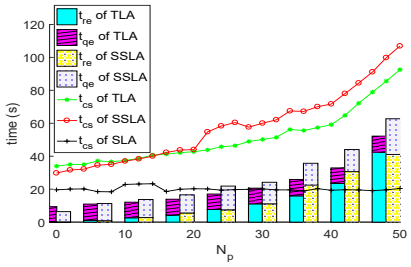


Figure 6: t_{cs}, t_{qe} and t_{re} VS N_p in *full* case

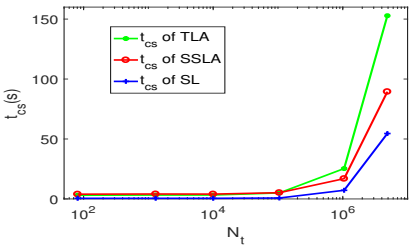


Figure 7: t_{cs} VS N_t in *full* case (log scale in X-axis)

Discussion. Although the performance and size of citations in the synthetic experiments are not acceptable for extreme values of view mappings (N_v), predicates (N_p) and tuple number (N_t), Table 10 shows that for our realistic cases these values are all very small (N_v and N_p are less than 10 while N_t is less than 10^4). Revisiting the results of Section 5.2, Figures 4a and 4b show that, in the *full* case, when N_v and N_p are less than 10, the performance and size of citations are very reasonable for all three approaches: t_{cs} is less than 1 minute, the citation generation time t_{cg} and the covering set size N_{cs} are also very small. However, since N_t is usually less than 10^4 whereas the synthetic workloads generate 10^6 tuples, t_{cs} for all three approaches in practice should be far less than 1 minute.

6 CONCLUSIONS

This paper builds on the notion of citation views [12] 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, and that reasoning at the schema level results in a 2-3x performance gain at the expense of generating citations to individual tuples.

The methods we propose are a first step towards fine-grained data bibliometrics, but they need to be integrated within a larger data citation infrastructure. Currently, none of the largest citation-based systems consistently take into account scientific datasets as targeted objects for use in academic work. In the future, the scientific community must define a theory of data citation which targets the problem of how to aggregate academic credit, and define appropriate impact measures.

In future work, we will explore the connection of citation to *provenance*. Since both provenance and citations are annotations on tuples, it may be possible to reason over provenance polynomials [18] of view definition tuples and 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 will explore how citations can be integrated into *data science environments*, in which queries are interleaved with analysis steps. We would also like to test (and possibly extend) our approach using other types of common datasets in this environment, e.g. dataframes, CSV, and time-series data.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. The work was partially funded by NSF IIS 1302212, NSF ACI 1547360, NIH 3-U01-EB-020954-02S1, and the CDC STARS-StG project of the University of Padua.

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] DataCite Metadata Schema Documentation for the Publication and Citation of Research Data, v4.0. Technical Report, DataCite Metadata Working Group, 2016.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] F. N. Afrati, C. Li, and J. D. Ullman. Using views to generate efficient evaluation plans for queries. *Journal of Computer and System Sciences*, 73(5):703 – 724, 2007.

[5] 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.

[6] A. Alawini, S. B. Davidson, W. Hu, and Y. Wu. Automating data citation in CiteDB. *PVLDB*, 10(12):1881–1884, 2017.

[7] R. Angles and C. Gutierrez. The Expressive Power of SPARQL. In *Proc. of the 7th International Semantic Web Conference (ISWC)*, pages 114–129, 2008.

[8] J. Brase, I. Sens, and M. Lautenschlager. The Tenth Anniversary of Assigning DOI Names to Scientific Data and a Five Year History of DataCite. *D-Lib Magazine*, 21(1/2), 2015.

[9] 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.

[10] P. Buneman and G. Silvello. A Rule-Based Citation System for Structured and Evolving Datasets. *IEEE Data Eng. Bull.*, 33(3):33–41, 2010.

[11] B. Cautis, A. Deutsch, and N. Onose. XPath rewriting using multiple views: Achieving completeness and efficiency. In *11th International Workshop on the Web and Databases, WebDB 2008, Vancouver, BC, Canada, June 13, 2008*, 2008.

[12] 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.

[13] A. Deutsch and V. Tannen. XML queries and constraints, containment and reformulation. *Theor. Comput. Sci.*, 336(1):57–87, 2005.

[14] W. Fan, X. Wang, and Y. Wu. Answering pattern queries using views. *IEEE Transactions on Knowledge and Data Engineering*, 28(2):326–341, Feb 2016.

[15] M. Force, N. Robinson, M. Matthews, D. Auld, and M. Boletta. Research Data in Journals and Repositories in the Web of Science: Developments and Recommendations. *Bulletin of IEEE Technical Committee on Digital Libraries, Special Issue on Data Citation*, 12(1):27–30, May 2016.

[16] FORCE-11. *Data Citation Synthesis Group: Joint Declaration of Data Citation Principles*. FORCE11, San Diego, CA, USA, 2014.

[17] 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.

[18] 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.

[19] P. Groth, A. Gibson, and J. Velterop. The Anatomy of a Nanopublication. *Inf. Serv. Use*, 30(1-2):51–56, 2010.

[20] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[21] S. Harding, J. Sharman, E. Faccenda, C. Southan, A. Pawson, S. Ireland, A. Gray, L. Bruce, S. Alexander, S. Anderton, C. Bryant, A. Davenport, C. Doerig, D. Fabbro, F. Levi-Schaffer, M. Spedding, and J. Davies. The IUPHAR/BPS Guide to PHARMACOLOGY in 2018: updates and expansion to encompass the new guide to IMMUNOPHARMACOLOGY. *Nucl. Acids Res.*, 46:D1091–D1106, 2018.

[22] L. B. Honor, C. Haselgrove, J. A. Frazier, and D. N. Kennedy. Data Citation in Neuroimaging: Proposed Best Practices for Data Identification and Attribution. *Frontiers in Neuroinformatics*, 10(34):1–12, August 2016.

[23] J. Klump, R. Huber, and M. Diepenbroek. DOI for Geoscience Data – How Early Practices Shape Present Perceptions. *Earth Science Inform.*, pages 1–14, 2015.

[24] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 655–664, New York, NY, USA, 2011. ACM.

[25] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 65–76, 2002.

[26] 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.

[27] S. Pröll and A. Rauber. A Scalable Framework for Dynamic Data Citation of Arbitrary Structured Data. In *Proc. of 3rd Int. Conf. on Data Management Technologies and Applications*, pages 223–230, 2014.

[28] 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.

[29] A. Rauber, A. Ari, D. van Uytvanck, and S. Pröll. Identification of Reproducible Subsets for Data Citation, Sharing and Re-Use. *Bulletin of IEEE Technical Committee on Digital Libraries, Special Issue on Data Citation*, 12(1):6–15, May 2016.

[30] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 551–562, 2004.

[31] G. Silvello. A Methodology for Citing Linked Open Data Subsets. *D-Lib Magazine*, 21(1/2), 2015.

[32] 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.

[33] N. Simons. Implementing DOIs for Research Data. *D-Lib Magazine*, 18(5/6), 2012.

[34] 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 the preprocessing step are shown in Algorithm 1.

Algorithm 1: Preprocessing step

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 $\mathcal{M}$, the extended query $Q_{ext1}(\bar{X}')$

1 Initialize $\mathcal{M} = \{\}$
2 Initialize the schema of the extended query $\bar{X}' = \bar{X}$
3 **for** *each view* $V \in \mathcal{V}$ **do**
4 | Derive all possible view mappings from V to Q that follows definition 3.1 and the last two conditions in definition 3.4 and add them to $\mathcal{M}$.
5 **end**
6 **for** *each view mapping* $M \in \mathcal{M}$ **do**
7 | Derive lambda terms $L(M)$ and the predicates $condition(M)$ under M
8 | Add all lambda terms in $L(M)$ to $\bar{X}'$
9 | Add boolean expressions of all the global conditions in $condition(M)$ to $\bar{X}''$
10 **end**
11 **for** *each relation* B_i *in the body of* Q **do**
12 | Add the view vectors $Vec(B_i)$ to $\bar{X}'$.
13 **end**
14 Construct the extended query Q_{ext1} with the following form:
15 $Q_{ext1}(\bar{X}') : -B_1(\bar{X}_1), B_2(\bar{X}_2), \ldots, B_m(\bar{X}_m), condition(Q)$
16 **return** $\mathcal{M}$, $Q_{ext1}(\bar{X}')$

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 2.

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 3. Algorithms 2 and 3 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*).

After the reasoning and the population steps, every tuple in $Q_{ext1}(D)$ is annotated with covering sets. The covering sets for the entire query are then calculated, which is similar to Algorithm 3 and thus not presented here. The aggregated covering sets are then

Algorithm 2: 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 $\mathcal{M}(t)$, a set of maximally covered relations $MCR(t)$

Initialize $\mathcal{M}(t) = \{\}$
/* $\mathcal{M}(t)$ denotes a set of valid view mapping */
for *each view vector* $Vec(B_i)(i = 1, 2, \ldots, m)$ **do**
  for *each annotated view* V *in* $Vec(B_i)$ **do**
    for *each view mapping* M *that the view* V *is involved in* **do**
      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 $\mathcal{M}(t)$
        add all the relations in Q that M covers into $MCR(t)$
      end
    end
  end
end
return $\mathcal{M}(t)$ and $MCR(t)$

converted to formatted citations using Algorithm 4. All the steps for TLA are shown in Algorithm 5.

Algorithm 3: Deriving the covering sets

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 $\mathcal{M}(t)$, a set of maximally covered relations $MCR(t)$
Output : A set of covering sets $C(t)$

Initialize $C(t) = \{\}$
for *each relation* $R(\bar{X}_R)$ *from* $MCR(t)$ **do**
  for *each variable* v *from* $\bar{X}_R \bigcap \bar{X}$ **do**
    find a set of view mappings $\mathcal{M}_\mathcal{R} \in \mathcal{M}(t)$ in which each view mapping can cover v
    if $\mathcal{M}_\mathcal{R}$ *is not* $\emptyset$ **then**
      $C(t) = cross\_product(C(t), \mathcal{M}_\mathcal{R})$
    end
  end
end
for *any two view mapping sets* C_i *and* C_j *in* $C(t)$ **do**
  if $C_i \subset C_j$ **then**
    remove C_i from $C(t)$
  end
end
return $C(t)$

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 6.

Algorithm 4: 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 $C(Q)$

Initiate $C(Q) = \{\}$
for *each group* $G \in \mathcal{G}$ **do**
  find all the view mappings $M(G)$ in some $AC \in AC(Q)$ so that any $M \in M(G)$ are valid for the group G
  Get the values of lambda terms for view mappings in $M(G)$ by using all the tuples in G
end
for *each covering set* $AC \in AC(Q)$ **do**
  Generate citation C using the values of lambda terms and the citation queries corresponding to the each view in AC.
  Put the citation C in $C(Q)$
end
remove duplicates in $C(Q)$
return $C(Q)$

Algorithm 5: 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: $Q(\bar{X}) : -B_1(\bar{X}_1), B_2(\bar{X}_2), \ldots, B_m(\bar{X_m}), condition(Q)$

Derive all the possible view mappings $\mathcal{M}$ and $Q_{ext1}(\bar{X'})$ using Algorithm 1
Execute query $Q_{ext1}(\bar{X'})$ in D:
Tuples in $Q_{ext1}(D)$ are grouped during the execution of $Q_{ext1}(\bar{X'})$. Suppose there are r groups, $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$, then derive covering sets for each group $G \in \mathcal{G}$ using Algorithm 3.
/* Aggregated covering sets */
$AC(Q) = \{\}$
if *Agg is Intersection* **then**
  find a set of view mappings $VM(Q)$ which are valid for all $t \in Q(D)$
  calculate $AC(Q)$ using view mappings from $VM(Q)$
end
if *Agg is Union* **then**
  find a set of view mappings $VM(Q)$ which are valid for at least one $t \in Q(D)$
  calculate $AC(Q)$ using view mappings from $VM(Q)$
end
Generate formatted citations for covering sets $AC(Q)$ using Algorithm 4.

After deriving the set of valid view mappings, we obtain the covering sets using Algorithm 3. Algorithms 6 and 3 form the *reasoning step* for SSLA.

Algorithm 6: 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 $MCR(t)$

Initialize $\mathcal{M}(t) = \{\}$
Initialize $MCR(t) = \{\}$
for *each view mapping* M *in* $\mathcal{M}$ **do**
  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
  if *M follows the rule above* **then**
    add M to $\mathcal{M}(t)$
  end
  add all the relations in Q that M covers into $MCR(t)$
end
return $\mathcal{M}(t)$ and $MCR(t)$

Algorithm 7: Details of SSLA

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 $\mathcal{M}$ and extended query $Q_{ext2}(\bar{X}')$ using Algorithm 1
2 Execute query $Q_{ext2}(\bar{X}')$ in D:
3 Tuples in $Q_{ext2}(D)$ are grouped during the execution of $Q_{ext2}(\bar{X}')$. Suppose there are r groups, $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$, then derive covering sets for each group $G \in \mathcal{G}$ using Algorithm 3.
/* Aggregated covering sets */
4 $AC(Q) = \{\}$
5 **if** *Agg is Intersection* **then**
6 find a set of view mappings $VM(Q)$ which are valid for all $t \in Q(D)$
7 calculate $AC(Q)$ using view mappings from $VM(Q)$
8 **end**
9 **if** *Agg is Union* **then**
10 find a set of view mappings $VM(Q)$ which are valid for at least one $t \in Q(D)$
11 calculate $AC(Q)$ using view mappings from $VM(Q)$
12 **end**
13 Generate formatted citations for covering sets $AC(Q)$ using Algorithm 4.

The *population*, *aggregation* and *citation generation* steps in SSLA are the same as in TLA. All the steps for SSLA are shown in Algorithm 7.

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 3, 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 1 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 [34] 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 6, 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 8

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.

Algorithm 8: 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 $\frac{cost(M)}{|Hv(M) \cap MCH(t)| + |R(M) \cap MCR(t)|}$
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** $C(t)$

In Algorithm 9 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 sub-query 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 [17], 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.

Algorithm 9: 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
1 $C \leftarrow \emptyset$;
2 $T_Q \leftarrow$ get table set of Q;
3 **foreach** $V_i \in \mathcal{V}$ **do**
4 /* Assign the weight to the views */;
5 $w_i \leftarrow |T_{V_i} \setminus T_Q| + 1$;
6 **end**
7 **while** $T_Q \neq \emptyset$ *and* $\mathcal{V} \neq \emptyset$ **do**
8 **foreach** $V_i \in \mathcal{V}$ **do**
9 /* Assign the cost to the view */;
10 $c_i = \frac{w_i}{|\{C \cup \{T_{V_i} \cap T_Q\}\} \backslash C|}$;
11 **end**
12 Choose the set of views $\mathcal{V}_T$ sharing the minimum cost c;
13 $C_{tmp} \leftarrow \emptyset$;
14 **foreach** $V_i \in \mathcal{V}_T$ **do**
15 $Q_t \leftarrow$ `getSubquery`($\mathcal{L}_Q, T_{V_i}$);
16 **if** `GL01`(V_i, Q_t) **then**
17 $C_{tmp} \leftarrow C_{tmp} \cup V_i$;
18 **else**
19 $\mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
20 **end**
21 **end**
22 **if** $|C_{tmp}| > 1$ **then**
23 Choose the view $V_i \in C_{tmp}$ minimizing $|\bar{Y}_i \setminus \bar{X}| + |\lambda_{V_i} \triangle \lambda_Q|$;
24 $C \leftarrow C \cup V_i, T_Q \leftarrow T_Q \setminus T_{V_i}, \mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
25 **else**
26 $C \leftarrow C \cup V_i, T_Q \leftarrow T_Q \setminus T_{V_i}, \mathcal{V} \leftarrow \mathcal{V} \setminus V_i$;
27 **end**
28 **end**
29 **return** C

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:

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

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

$\lambda pk.V1(pk, pt, pconf) : -dblp\_paper(pk, pt, pyear, pconf)$
$\lambda ck.V2(ck, cn, detail) : -dblp\_conference(ck, cn, detail)$
$\lambda py, cn.V3(pk, pt, py, cn) : -dblp\_paper(pk, pt, py, pconf), dblp\_conference(ck, cn, detail), cn = pconf$
$\lambda aid.V4(aid, award, abs, amt, eff\_date, ex\_date) : -grants\_awards(aid, award, eff\_date, ex\_date, amt, instr, org, officer, abs, \dots)$
$\lambda eff\_date, org.V5(aid, award, abs, amt, eff\_date, ex\_date) : -grants\_awards(aid, award, eff\_date, ex\_date, amt, instr, org, officer, abs, \dots)$
$\lambda pk.V6(pk, pt, pconf) : -dblp\_paper(pk, pt, pyear, pconf)$

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

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

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:

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

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

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