Preference-aware Integration of Temporal Data

Bogdan Alexe
IBM Almaden
balexe@us.ibm.com

Mary Roth
IBM Almaden and UCSC
torkroth@us.ibm.com

Wang-Chiew Tan
UCSC
tan@cs.ucsc.edu

ABSTRACT

A complete description of an entity is rarely contained in a single data source, but rather, it is often distributed across different data sources. Applications based on personal electronic health records, sentiment analysis, and financial records all illustrate that significant value can be derived from integrated, consistent, and queryable profiles of entities from different sources. Even more so, such integrated profiles are considerably enhanced if temporal information from different sources is carefully accounted for.

We develop a simple and yet versatile operator, called PRAWN, that is typically called as a final step of an entity integration workflow. PRAWN is capable of consistently integrating and resolving temporal conflicts in data that may contain multiple dimensions of time based on a set of preference rules specified by a user (hence the name PRAWN for *preference-aware union*). In the event that not all conflicts can be resolved through preferences, one can enumerate each possible consistent interpretation of the result returned by PRAWN at a given time point through a polynomial-delay algorithm. In addition to providing algorithms for implementing PRAWN, we study and establish several desirable properties of PRAWN. First, PRAWN produces the same temporally integrated outcome, modulo representation of time, regardless of the order in which data sources are integrated. Second, PRAWN can be customized to integrate temporal data for different applications by specifying application-specific preference rules. Third, we show experimentally that our implementation of PRAWN is feasible on both "small" and "big" data platforms in that it is efficient in both storage and execution time. Finally, we demonstrate a fundamental advantage of PRAWN: we illustrate that standard query languages can be immediately used to pose useful temporal queries over the integrated and resolved entity repository.

1. INTRODUCTION

Complete information about an entity is rarely contained in a single data source, but rather, it is often distributed across different data sources. As a result, there is great value in combining data from multiple sources to build a comprehensive understanding of entities. The combined understanding of entities is considerably enhanced if temporal information from different sources is also carefully accounted for [31, 34] in order to determine when facts about entities are true.

For example, patients typically visit multiple medical professionals/facilities over the course of their lifetime, and often even simultaneously. While it is important for each medical facility to maintain medical history records for its patients to provide more comprehensive diagnosis and care, there is even greater value for both the patient and the medical professionals to have access to an integrated profile derived from the histories kept by each institution. Through the integrated profile, one could understand *when* a drug was administered and taken by a patient and for how long. In turn, one could determine whether drugs with adverse interactions have been unintentionally prescribed to a patient by different institutions at the same time. Another example comes from the retail industry, where retailers are interested to understand the profile, purchase intents, and sentiments of their (potential) customers over time. With a comprehensive understanding of an individual drawn from different sources, including *when* certain intentions or sentiments are expressed, recommendations and advertisements can be targeted appropriately. Yet another real world example comes from reports that are filed with the U.S. Securities and Exchange Commission (SEC) at different times, where there is a critical need to provide an integrated understanding across individual reports of the stock holdings and professional relationships of executives over time (which we will detail shortly). These examples all illustrate that the time aspects of data can be critically relevant and, in particular, it is important to know the time periods in which a fact about an entity is true.

Several challenges arise when integrating *temporal data*, which refers to data that contains explicit time-specific information, such as the date of a prescription, or implicit time information, such as the version number or timestamp of an instance. First, the time aspect associated with the data is often *imprecise*. A facility may report that a patient was treated for a condition on a specific date. From this information, we can infer that the patient must have had the condition on the day he was seen, but we cannot say if the patient still has the condition, or for how long prior to or after the visit he had the condition.

Second, as in traditional data integration, *inconsistencies* may arise with respect to certain constraints when data from multiple sources are combined together. In our setting, an added complexity arises from the need to handle certain constraints across time [24]. For example, reports that are filed with the SEC or corporate press releases may state that an executive held a particular title on a given day, but it does not provide information about when that title was first held, or even if it is still held after the report or press release is made public. Another data source (or even the same data source

Proceedings of the VLDB Endowment, Vol. 8, No. 4

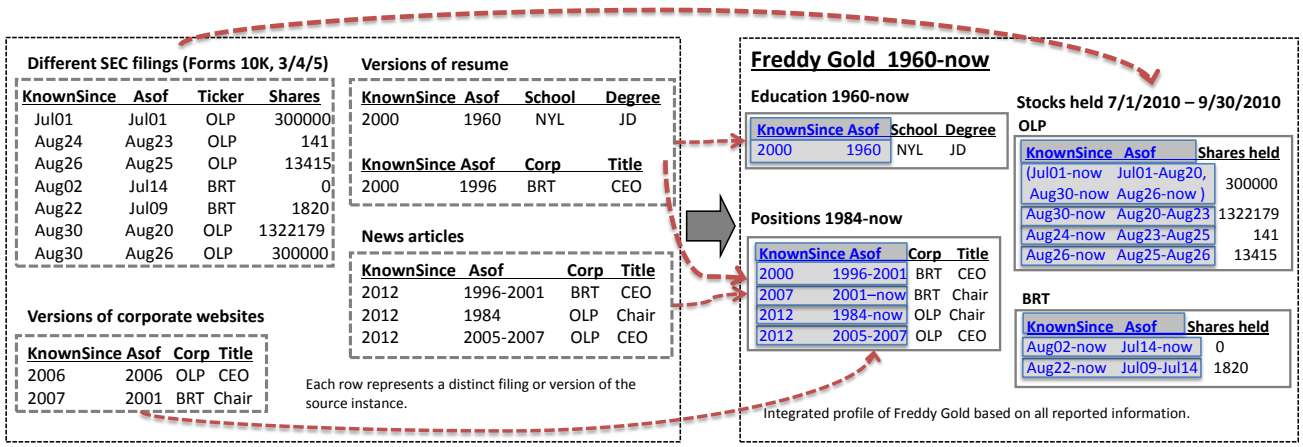

Figure 1: Creating an integrated profile of Freddy Gold from multiple temporal sources. Temporal contexts are shaded (in blue).

at a different time) may report that the executive was employed by the company at a later date and with a different title. If there is a constraint that an employee can only have a single title at any point in time, what can we infer about the employment history of the executive? Should we assume that she had been employed by the company as of the (earlier) date associated with her title for some time before it was changed to the other title at a later date, or should the earlier title be completely disregarded? How would the integration be different if we had assumed that an employee can hold multiple titles at any point in time or if we had preferred information from one source over the other?

We illustrate next with a concrete example the subtleties that are associated with consistently integrating temporal data.

Motivating Example Figure 1 shows a simplified form of a real example where information about Freddy Gold is obtained and integrated from several sources and at various times. The data sources include reports that are continually filed with the SEC (Forms 10K and 3/4/5) and are available via the EDGAR database [15], and different versions of resumes, corporate websites, and news articles available electronically. However, for simplicity, our discussion will focus on the integration of SEC reports only. We assume that each row shown on the left of Figure 1 represents a separate filing or a version, even though in general, a filing or version may contain many rows of data.

For example, "Different SEC filings" in Figure 1 shows 7 facts from 7 reports filed with the SEC that each indicate the number of shares of a particular stock (OLP and BRT) held by Freddy Gold during the second half of 2010 (year is not shown). The first row shows that Freddy owned 300000 OLP shares on Jul01 in a report filed on Jul01. The second row shows Freddy owned 141 OLP shares on Aug23 in a report filed on Aug24. The third row shows Freddy had 13415 OLP shares on Aug25 in a report filed on Aug26.

How can we best reconcile and compactly represent the given information to understand, for example, what Freddy's affiliation was or how many shares of BRT he owned on Jul15? The 4th filing, with an'asof' date of Jul14, shows that he had 0 BRT shares on Jul14, and so it would be reasonable to assume that Freddy still had 0 shares of BRT on Jul15. On the other hand, the 5th filing has a 'knownsince' date of Aug22, which is later than the 'knownsince' date of the 4th filing, and it shows that he had 1820 shares as of Jul09, so another interpretation is that the 5th filing superscedes or corrects the 4th, and therefore Freddy had 1820 shares on Jul15. The correct interpretation depends the semantics of the application, but it should *not* depend upon the order in which the information is received and integrated. We dive into some of the details of how we can answer these questions next.

Overview of our approach Though the 'asof' date associated with a filing only records the day on which a fact was known to be true, it is reasonable to assume that the data in the filing *continues to be true until new information is received*. For example, the first SEC filing indicates that until we receive new information, Freddy owns 300000 OLP shares during the time interval Jul01-now and this was known since Jul01, which we denote with the same time interval Jul01-now. The second filing indicates that until we receive new information, Freddy owns 141 OLP shares during the time interval Aug23-now and this was known since Aug26, which we denote with another time interval Aug26-now.

Since there can be only one quantity of OLP shares owned by Freddy *at any point in time*, the two filings provide conflicting information on the quantity of shares from Aug23 onwards. An aggregated understanding can be derived based on the following *preference*: information with a *later* 'asof' date is preferred over information with an *earlier* 'asof' date. In this way, one can conclude that Freddy owns 300000 shares between Jul01-Aug23 and he owns 141 shares between Aug23-now according to the filings that were received on Jul01 and Aug24. This example describes a key step behind our *preference-aware union* operator: When the second SEC filing is integrated with the first, the asof time period for 300000 shares is retroactively adjusted from Jul01-now to Jul01-Aug23 to resolve conflicts. Our operator resolves temporal conflicts under constraints and preference rules that are provided by the administrator, if available. The result that is obtained does not depend on the order in which the data is integrated.

The aggregated knowledge from the entire set of SEC filings under the above preference is compactly represented in the upper portion of Figure 2. This outcome is obtained if filings are received and integrated in any order and one favors information with a later 'asof' time whenever there is a conflict. Other interpretations are possible; the bottom portion of Figure 2 shows the aggregated knowledge when it is assumed that later reports (i.e., later 'knownsince' date) always contain the latest information and hence, preferred over filings with an earlier 'knownsince' date. Under this interpretation, the 6th report (1322179 OLP shares) completely overwrites information from the 2nd report (141 OLP shares) and the 3rd report (13415 OLP shares).

In the above example, it happened that all conflicts are resolved based on preferences. If some conflicts cannot be resolved (e.g.,

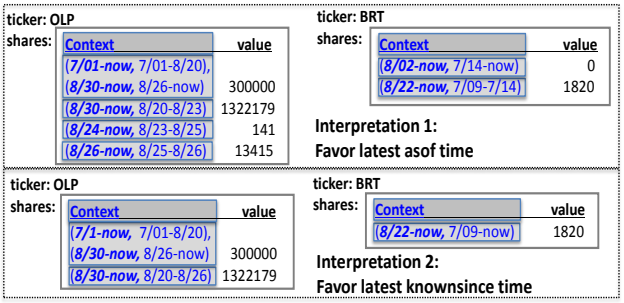

Figure 2: Two different ways of adjusting 'asof' time.

this may happen when no preferences are specified), then all conflicting values over time are retained.

The end-to-end entity integration framework PRAWN is a new operator that performs temporal integration as part of a traditional rule-based entity integration framework in which the goal is to produce a repository of clean, consistent entities that are aggregated from partial information available from a myriad of sources [1] and over which *consistent temporal query answers* can be easily derived. Figure 3 shows where PRAWN fits into the workflow of this framework. As shown in the figure, an administrator specifies rules at design time to customize the workflow for a particular application, and these rules are applied at runtime. Initial steps in the workflow include information extraction to extract structured information about the entity from unstructured sources (such as public or enterprise documents, social media feeds), schema mapping to map data extracted from both structured and unstructured sources to a common schema, and entity resolution to group data that corresponds to the same entity together in a unified object. PRAWN is typically invoked after the entity resolution step to reconcile temporal conflicts among an entity's attributes to produce a temporally consistent object.

In real-life scenarios where information is integrated from heterogeneous sources, the ability of PRAWN to resolve temporal conflicts according to multiple policies is especially important. Consider the scenario of creating the profile of a company executive across regulatory filings and social media activity. For an attribute such as number of shares held in a particular stock, the temporal resolution may prefer records with a later 'asof' value. In contrast, for an attribute such as current location, integrated from a social media feed, temporal resolution may take into consideration the provenance of the attribute, e.g. prefer the location inferred from geo-coordinates associated with a posted message over the information given by the user when the profile was created.

Other strategies are possible to derive a temporally consistent object, such as machine-learning based approaches for learning the true values among conflicting values (e.g., see [12, 30]). Both rule-based approaches and machine-learning based approaches are desirable and complementary approaches for managing conflicting values. However, the integrated outcome from rule-based approaches is more amenable to debugging and understanding in general. On the other hand, machine-learning based approaches allow one to determine a true value when no rules are known or can be learned. We refer the interested reader to [9] for an analogous discussion of rule-based and machine-learning based approaches in the context of information extraction systems. PRAWN is a rule-based operator where preference rules are used to discern preferred values. The rules can be used to understand how an integrated result is arrived at and values that are not known to be false will always be kept until more rules can be learned or applied, or machine-learning

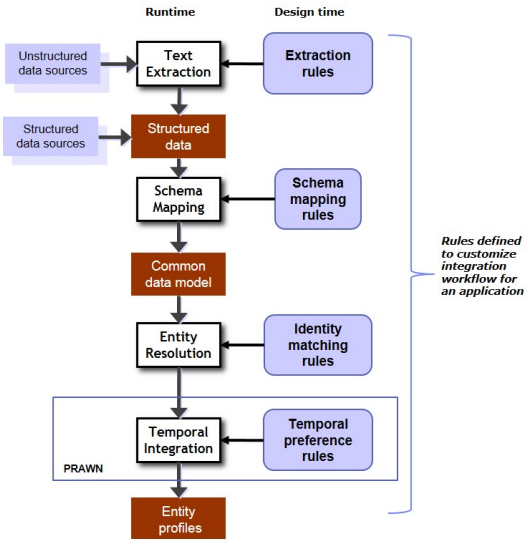

Figure 3: PRAWN enables temporal integration as part of an entity integration workflow.

approaches can also be applied to learn the true values among the preferred values.

Contributions We make the following contributions:

- We present a simple and yet versatile operator, called *preference-aware union* (PRAWN), that is capable of consistently merging (hierarchical) temporal data based on a set of preference rules specified by a user. To the best of our knowledge, this is the first operator for merging temporal data based on user-specified preference rules.
- In the event that not all conflicts can be resolved through preferences, one can enumerate each possible consistent interpretation of the result returned by PRAWN at a given time point through a polynomial-delay algorithm. We establish the connection between our work and prior work on *inconsistent database and repairs* and show that each consistent interpretation corresponds to a *repair* of the result at that given time point.
- We present several desirable properties of PRAWN.
  - We show that PRAWN upholds important algebraic identities, which makes it suitable for integrating temporal data: PRAWN produces the same integrated archive, modulo representation of time, regardless of the order in which sources are integrated.
  - We demonstrate the generality of PRAWN by illustrating how PRAWN can capture different classes of applications such as archiving scientific data, updates in bitemporal databases, integrating SEC data, and integrating (non-temporal) data.
  - We implemented the PRAWN operator and showcase the effectiveness of both a serial and parallel version of the operator experimentally on several real-world data sets. Specifically, we show that our implementations on both "small" and "big" data platforms are efficient in both storage and execution time.
  - Because our hierarchical data model captures both time and data as first-class citizens and can be represented easily in standard hierarchical formats such as XML or JSON, we show how the result of PRAWN is immediately admissible to temporal querying through standard query languages.

2. PRELIMINARIES

Data model We adopt a hierarchical data model for naturally capturing real-world data sources such as SEC, Twitter, and scientific databases, which typically occur in XML or JSON formats. Our hierarchical data model is based on nested relations. However, for simplicity, we will describe our data model through examples, which are largely relational.

Concepts related to time We now describe some concepts related to time: time instant, time interval, time dimension, time point, and temporal context.

We view time as a linear structure $(\mathbf{T}, <)$, where $\mathbf{T}$ is a discrete set of *time instants* and $<$ is a precedence relation on $\mathbf{T}$.

In this paper, we assume time are dates of the form `mmmdd` (e.g., Jun06), and a special symbol now denotes the current date. The pair [Time-Time) describes a *closed-open time interval*, which finitely represents a possibly infinite set of time instants. For example, [Jul01-Aug25) is a time interval that represents a finite number of days beginning on July 1, and ends *before* Aug 25, and [Jul01-now) represents an arbitrary number of days beginning on July 1.

There are multiple types of time and each type of time is called a *temporal dimension* [10]. There are two temporal dimensions, 'asof' and 'knownsince', in our running example. In bi-temporal databases, *valid* and *transaction* time are used. Sometimes a third dimension, called *decision time* [29], is also used along with valid and transaction time. As another example, it is conceivable that SEC data can be modeled with three time dimensions, with 'published' time in addition to 'asof' and 'knownsince', where 'published' time refers to the time when SEC made the filing public, and 'knownsince' refers to the time a particular quantity of shares with a particular 'asof' time is known to be true.

If there are n time dimensions, then a *time point* is an n-tuple $(t_1, ..., t_n)$, where t_i, $1 \leq i \leq n$, is a time instant in the ith temporal dimension. We write $\bar{t}$ to denote the vector $(t_1, ..., t_n)$ of time instants.

A *temporal context* is a set of records that describe a set of multi-dimensional time points. For example,

knownsince	asof
[Jul01-now)	[Jul01-Aug20)
[Aug30-now)	[Aug26-now)

is a set of two records, where each record describes a possibly infinite set of two dimensional time points. For example, ⟨Jul01,Jul01⟩ and ⟨Aug30,Aug26⟩ are time points that occur in the first record, and, respectively, the second record above, while the time point ⟨Jul01,Apr01⟩ does not occur in the above temporal context.

Intuitively, if a temporal context is associated with an entity, then the values of that entity's attributes are true during the set of all time points given by the temporal context. In what follows, we will abbreviate a temporal context with the notation ■ and describe the temporal context explicitly only when needed. A temporal context will always be shown shaded (in blue in color printout).

Prawn schemas and prawn instances A PRAWN schema is a set of relation schemas, where every non-key attribute may contain a set of values annotated with temporal contexts. We will denote the usual schema with $\mathbf{S}$ and the corresponding PRAWN schema as $\mathbf{S_p}$

$\mathbf{S}$: Stock(ticker*:Str,shares:Int).

$\mathbf{S_p}$: stocksHeld::= Stock(ticker*:Str, shares:SetOfPreferred[■Int])

In the above $\mathbf{S}$ denotes the Stock relation (without time) and the corresponding PRAWN schema is $\mathbf{S_p}$. The notation "*" denotes the key attribute. Under $\mathbf{S_p}$, the non-key attribute 'shares' may contain a set of (conflicting) values. The set of conflicting values is flagged with a special `SetOfPreferred` type to denote that this is a set of values that remain after preference rules (see Section 3) have been applied. (Hence the name "`SetOfPreferred`".) Each value in this set is associated with a temporal context ■ that specifies the set of all time points when the value is true. In other words, the temporal contexts capture *when* various quantities of shares of a stock exist. In general, temporal contexts can also be associated with the set or the relation but we have omitted them for simplicity.

In effect, our data model extends the deterministic data model of [7], which is used in [8]. However, the annotation scheme of [8] is limited to describing a single time interval and the algorithm of [8] makes an implicit hard-coded preference for information in the later version. As we shall explain, our algorithm allows more than one time dimension and provides a declarative mechanism for domain experts to specify preferences.

The annotated table below is a PRAWN instance that conforms to $\mathbf{S_p}$. It shows that for OLP, 300000 shares is the preferred true value of the number of shares at some time point while 141 is the preferred true value of the number of shares at some other time points.

ticker	shares	
OLP	knownsince: [Jul01-now), [Aug30-now) / asof: [Jul01-Aug20), [Aug26-now)	300000
	knownsince: [Aug24-now) / asof: [Aug23-Aug25)	141

Snapshot We write $D@\langle t_1, ..., t_n\rangle$, written as $D@\bar{t}$ in short, to denote *the snapshot of D at time point $\bar{t}$*. Without belaboring the exact definition of a snapshot, which is defined inductively on the structure of a schema, we give an example next. If D denotes the instance above, then the respective snapshots $D@\langle$Jul01,Jul01$\rangle$ and $D@\langle$Aug24,Aug23$\rangle$ are shown below.

ticker	shares
OLP	300000

ticker	shares
OLP	141

Each snapshot above consists of only one fact. As another example, the snapshot $D@\langle$Jul01,Aug22$\rangle$ is the emptyset, since the time point ⟨Jul01,Aug22⟩ is not among the set of time points represented by the temporal contexts in D.

Conformance to schema and key constraints Let $\mathbf{S}$ be a schema and $\mathbf{S_p}$ denote the corresponding PRAWN schema. We say that an instance D *conforms* to $\mathbf{S_p}$ if at every time point $\bar{t}$, the snapshot $D@\bar{t}$ conforms to the schema $\mathbf{S}$ in the usual sense. Similarly, we say that D *satisfies a key constraint* K (i.e., D is *consistent* w.r.t. to K), if for every time point $\bar{t}$, the snapshot $D@\bar{t}$ satisfies K. In other words, apart from the structural and type constraints of $\mathbf{S}$ that must be adhered to, every relation in $D@\bar{t}$ must satisfy the associated key constraint.

In the full generality of our data model, we require that there is a key defined for every (nested) relation. This enables every element of an instance to be uniquely identified by a sequence of element names and key values along the *path* from the root of the instance to that element.

PROPOSITION 2.1. *Let D be an instance of a* PRAWN *schema and $\bar{t}$ be a time point. Then, every element (i.e., set or record or atomic value) of $D@\bar{t}$ can be uniquely identified by some path from the root of $D@\bar{t}$ to that element.*

A path has the form $D@\bar{t}/\langle p\rangle$ or $D/\langle p\rangle$, where D is an instance, $\bar{t}$ is a time point, and $\langle p\rangle$ is a path in $D@\bar{t}$. Sometimes D is omitted if it is understood from the context. For example, the path $D@\langle$Jul01,Jul01$\rangle$/[ticker=OLP] uniquely identifies the OLP record in the stocksHeld relation on ⟨Jul01,Jul01⟩ and the set of

quantities of shares of stock OLP on ⟨Jul01,Jul01⟩ is identified by the path $D@\langle$Jul01,Jul01$\rangle$/[ticker=OLP]/shares.

3. PREFERENCE-AWARE UNION (PRAWN)

Recall that the PRAWN operator is part of an entity integration framework. It assumes that data from different sources or different versions are already transformed into a common PRAWN schema before the PRAWN operator is applied. PRAWN is a binary operator that derives a merged instance from two instances of the same schema based on a set of preference rules. We describe how preference rules are declaratively specified next.

3.1 Preference rules

A *preference rule* may be specified for each non-key attribute. For example, one may state that values of shares with a later 'asof' date are preferred over values with an earlier 'asof' date as follows:

ψ_1: for r_1 and r_2 in stocksheld/Stock[ticker=*]/shares
prefer r_1 if r_1.start(asof) > r_2.start(asof)

The path 'stocksheld/Stock[ticker=*]/shares' refers to the set of values under the attribute shares of each tuple in the Stock relation. It states that if there are two quantities of shares that coexist at some time point, preference will be given to the quantity that is associated with a greater beginning 'asof' time.

As another example, suppose there are three data sources A, B, and C. The preference that data about quantities of shares from source A is preferred over that of source B can be specified follows:

ψ_2: for r_1 and r_2 in stocksheld/Stock[ticker=*]/shares
prefer r_1 if r_1.source = 'A' and r_2.source = 'B'

In the above, the metadata 'source' describes the origin. The rule states that share values from source A is preferred over those from source B but it does does not mention preferences over data from B and C, or from A and C for that matter.

In general, a preference rule ψ is of the following form:

ψ_{path}: for r_1 and r_2 in ⟨*path*⟩
prefer r_1 if $\phi(r_1, r_2)$

where $\phi(r_1, r_2)$ is a conjunction of comparison predicates of the form $v_1\,\theta\,v_2$, where v_1 is either an attribute of r_1 or a constant, v_2 is either an attribute of r_2 or a constant, and θ is one of the comparison operators $\{<, \leq, >, \geq, =, <>\}$.

Every preference rule over an instance D induces a *preference relation* ψ^D_{path}, which describes an explicit partial ordering between values. We assume preference relations are acyclic binary relations that are also closed under transitivity. Otherwise, one can extract an acyclic binary relation as follows: Take the transitive closure of the relation. If r is preferred over r' and r' is not preferred over r, then the new relation asserts that r is preferred over r'. In other words, cyclic preferences are omitted in the new relation.

We say that *an instance D satisfies a preference rule ψ_{path}* if for every time point $\bar{t}$, there does not exist distinct r_1 and r_2 that both occur in the set referred to by path ⟨*path*⟩ in $D@\bar{t}$ such that $\phi(r_1, r_2)$ holds. A *preference specification* is a set of preference rules. *An instance D satisfies a preference specification* Ψ if D satsifies every preference rule $\psi \in \Psi$.

DEFINITION 3.1. (Preference over PRAWN instances) *Given two instances D and D' of the same schema $\mathbf{S_p}$ and a preference specification Ψ over $\mathbf{S_p}$, we say D'* is preferred over *D w.r.t. Ψ, denoted as $D' \gg_\Psi D$, if*

1. *D' satisfies Ψ, D satisfies Ψ, and*
2. *for every time point $\bar{t}$, it is the case that $D'@\bar{t}$ dominates $D@\bar{t}$.*

We say that $D'@\bar{t}$ dominates *$D@\bar{t}$ if for every path p in $D@\bar{t}$,*

- *p also occurs in $D'@\bar{t}$, and*
- *if p points to a set, it must be that for every element $e \in (D@\bar{t}/p - D'@\bar{t}/p)$, there is an element $e' \in D'@\bar{t}/p$ such that e' is preferred over e according to Ψ.*

Intuitively, if $D'@\bar{t}$ dominates $D@\bar{t}$, then every element in $D@\bar{t}/p$ either occurs in $D'@\bar{t}/p$ or is dominated by some element in $D'@\bar{t}/p$.

When combining information from different sources or different versions of the same source, our goal is to derive a "maximal" instance that satisfies the given preference rules. We formalize the meaning of such an instance next.

DEFINITION 3.2. (Preference-aware union $\uplus_\Psi$) *Let D_1 and D_2 be two instances of a* PRAWN *schema $\mathbf{S_p}$ and Ψ be a preference specification. Then $D_1 \uplus_\Psi D_2$ is an instance D of $\mathbf{S_p}$ such that*

1. *D satisfies Ψ,*
2. *for every time point $\bar{t}$ and for every path p in $D@\bar{t}$, it must be that p is a path in $D_1@\bar{t}$ or $D_2@\bar{t}$,*
3. *for every other instance D' such that the above holds, it is not the case that $D' \gg_\Psi D$.*

The first condition states that D satisfies the preference rules. The second condition ensures that values of D are from D_1 or D_2 and the third condition ensures that D is "maximal".

3.2 Properties of the Prawn operator

We first show that there is a unique instance that represents the merge $D_1 \uplus_\Psi D_2$.

PROPOSITION 3.1. *Let D_1 and D_2 be two instances of a* PRAWN *schema $\mathbf{S_p}$ and Ψ be a preference specification. Then, there is a unique instance of $\mathbf{S_p}$ (modulo representation of time points) for $D_1 \uplus_\Psi D_2$.*

Next, we show that $\uplus_\Psi$ is idempotent, commutative, and associative. The idempotence property ensures that an instance that is integrated with itself will continue to be the same as the original instance. Furthermore, with commutativity and associativity, we obtain the following guarantee: regardless of the order that data sources are integrated, though the syntactical representation of time instants may vary, the final result contains identical time points. This operator is thus well-suited for entity integration as data from different sources (and the same source) can be integrated in any order, as and when it becomes available.

The proof of Theorem 3.2 (see below) can be found in [2] and it makes use of the definition of equivalence, which is defined next. The proof of idempotency is straightforward since the inputs to $\uplus_\Psi$ are identical and already satisfy Ψ, and $D \uplus_\Psi D$ is exactly D by Definition 3.2. We prove commtuativity by showing that at every time instant $\bar{t}$, $(D_1 \uplus_\Psi D_2)@\bar{t}$ is identical to $(D_2 \uplus_\Psi D_1)@\bar{t}$. We prove associativity by showing that at every time instant $\bar{t}$, $((D_1 \uplus_\Psi D_2) \uplus_\Psi D_3)@\bar{t}$ is identical to $(D_1 \uplus_\Psi (D_2 \uplus_\Psi D_3))@\bar{t}$, and then apply induction on the structure of $\mathbf{S}$.

DEFINITION 3.3. Let D_1 and D_2 be two instances that conform to a PRAWN schema $\mathbf{S_p}$. We say that D_1 is *equivalent* to D_2, denoted as $D_1 \equiv D_2$, if for every time point $\bar{t}$, the instance $D_1@\bar{t}$ is *identical* to $D_2@\bar{t}$ in the following sense:

- if $D_1@\bar{t}$ and $D_2@\bar{t}$ are sets of atomic values, then the sets D_1 and D_2 are equal. That is, $D_1 = D_2$.
- if $D_1@\bar{t}$ and $D_2@\bar{t}$ refer to sets of (non-atomic) elements, then $e \in D_1@\bar{t}$ iff $e' \in D_2@\bar{t}$ where e and e' have the same key value, and e is identical to e'.

- if $D_1@\bar{t}$ and $D_2@\bar{t}$ are records (i.e., tuples of relations) of the form Rcd$[l_1 : v_1, \ldots, l_k : v_k, ...]$ and, respectively, Rcd$[l_l : v'_1, \ldots, l_k : v'_k, ...]$, then v_1 is identical to v'_1, ..., and v_k is identical to v'_k.

THEOREM 3.2. *Let D_1, D_2, and D_3 be instances that conform to the same* PRAWN *schema $\mathbf{S_p}$, and let Ψ be a set of preference specifications such that D_1 (resp. D_2 and D_3) satisfies Ψ. Then, the following identities hold for $\uplus_\Psi$:*

- *(Idempotence)* $(D_1 \uplus_\Psi D_1) \equiv D_1$.
- *(Commutativity)* $D_1 \uplus_\Psi D_2 \equiv D_2 \uplus_\Psi D_1$.
- *(Associativity)* $(D_1 \uplus_\Psi D_2) \uplus_\Psi D_3 \equiv D_1 \uplus_\Psi (D_2 \uplus_\Psi D_3)$.

Connection to inconsistent databases and repairs In the event that not all conflicts can be resolved through preferences, one can enumerate each possible consistent interpretation of the result returned by PRAWN at a given time point through a polynomial-delay algorithm. We first describe the connection between our work and prior work on inconsistent database and repairs. We show that each consistent interpretation of the integrated result corresponds to a *repair* at that given time point. After this, we describe the polynomial-delay algorithm that enumerates each repair.

Recall that an *inconsistent database* [3] is a database that may violate some integrity constraints (e.g., keys). The fundamental philosophy behind inconsistent databases is to conservatively keep all inconsistencies and resolve inconsistencies through some other means or during query time (e.g., through the notion of *certain answers*). This is in contrast to applying data cleaning techniques, which is sometimes perceived as ad hoc, to derive a single consistent database.

The *integrated archive* $D_1 \uplus_\Psi D_2$ is, in general, an *inconsistent temporal database* as it captures, at every time point, the remaining inconsistencies at that time point after preferences have been applied. For example, suppose there are no preference rules for our SEC reports. If D_1 and D_2 refer to the first and second SEC filings under 'Different SEC reports' of Figure 1, then $D_1 \uplus_\Psi D_2$, shown below, is an inconsistent temporal database.

ticker	shares	
OLP	knownsince: [Jul01-now), asof: [Jul01-now)	300000
	knownsince: [Aug24-now), asof: [Aug23-now)	141

This is because the snapshot $(D_1 \uplus_\Psi D_2)@\bar{t}$, where $\bar{t}$ is the time point ⟨Aug24,Aug23⟩, is an inconsistent database, as it contains two quantities of shares (i.e., 300000 and 141) for OLP. This violates the key of the relation of $\mathbf{S}$ which asserts that there can only be one quantity of share for each ticker symbol at any time point. Similarly, $(D_1 \uplus_\Psi D_2)@\langle$Aug30,Aug30$\rangle$ and $(D_1 \uplus_\Psi D_2)@\langle$Dec1,Dec1$\rangle$ are inconsistent databases.

Let D be an instance of a PRAWN schema, and $\bar{t}$ be a time point. An instance D' is an *optimal repair* of $D@\bar{t}$ w.r.t. a preference specification Ψ and a set $\mathcal{K}$ of key constraints if

1. D' satisfies Ψ and $\mathcal{K}$.
2. for every path p in D', it must be that p is a path in $D@\bar{t}$, and
3. there does not exist another instance D'' such that the above holds for D'', and D'' dominates D'.

Intuitively, D' is a maximal consistent database of $D@\bar{t}$ that satisfies Ψ and $\mathcal{K}$. In particular, if $\Psi = \emptyset$ in the relational case, each maximal consistent database of $D@\bar{t}$ corresponds to a *subset repair* [3] of the inconsistent database $D@\bar{t}$ w.r.t. $\mathcal{K}$ (hence, the name *optimal repair*).

Continuing with our example, there are two optimal repairs at the time point ⟨Aug24,Aug23⟩, which are shown below:

ticker	shares
OLP	300000

ticker	shares
OLP	141

Given an instance D of $\mathbf{S_p}$ and a time point $\bar{t}$, there is a polynomial-delay algorithm that enumerates all the optimal repairs of D w.r.t. Ψ and $\mathcal{K}$ at time point $\bar{t}$.

Next, we informally describe a polynomial-delay algorithm for enumerating all optimal repairs of an instance D w.r.t. Ψ and $\mathcal{K}$ at time point $\bar{t}$. Given an input to a problem, a *polynomial-delay algorithm* [25] generates the first solution in polynomial time in the size of the input and each subsequent solution in polynomial time in the size of the input.

Polynomial-delay enumeration algorithm The enumeration algorithm keeps a pointer for every SetOfPreffered type in D. Initially, each pointer points to the first element of the set. To generate the first optimal repair, D is traversed in document-order. Whenever the traversal reaches a the path that refers to an element whose temporal context contains $\bar{t}$, the algorithm emits that element. Whenever a path refers to a SetOfPreferred type, the algorithm emits the next element e (according to the pointer) of that set whose temporal context contains $\bar{t}$ and advances the pointer. This check can be done in polynomial time in the size of the temporal context. To generate the next optimal repair, we repeat the procedure above.

The constructed output is a optimal repair because there can be at most one element per SetOfPreferred type under a key constraint. It is also easy to verify that each optimal repair can be constructed in polynomial time in the size of D and $\bar{t}$.

The integrated archive D returned by PRAWN is a compact representation of all optimal repairs over time. Note however that the integrated archive has the property that at every time point $\bar{t}$, $D@\bar{t}$ satisfies Ψ (since preference rules have already been applied by PRAWN) even though $D@\bar{t}$ may not satisfy $\mathcal{K}$. So in particular, this polynomial-delay algorithm gives us the option to systematically enumerate the remaining consistent interpretations (i.e., maximal consistent databases) of $D_1 \uplus_\Psi D_2$ in the event that not all conflicts can be resolved through preference rules.

3.3 The PRAWN Algorithm

We now describe the PRAWN algorithm that computes $D_1 \uplus_\Psi D_2$. Obviously, it is infeasible to resolve inconsistencies that may occur at every time point according to Definition 3.2, since there can be a large number of time points to consider in general. Our algorithm computes $D_1 \uplus_\Psi D_2$ by manipulating temporal contexts to satisfy the given preference specification.

The PRAWN algorithm recurses on the structure of D_1 (resp. D_2) based on the types of D_1 (resp. D_2). Observe that the elements of D_1 and D_2 must have the same type as the algorithm proceeds, since they both conform to the schema $\mathbf{S_p}$. Otherwise, an error is immediately returned (line 26).

For simplicity, we present our algorithm only for the relational case, where a relation is represented as a `SetOf` records, and each record is defined as `Rcd`$[l_1 : \tau_1, \ldots, l_n : \tau_n, ...]$, where l_i are labels (or attributes) and each corresponding τ_i is an atomic type `Str` or `Int`. The "..." at the end allows for flexibility in the definition of attributes in the relation. It denotes that the record may contain more and more attributes as integration occurs. An attribute of a record can also be a nested set of `Str` or `Int` values, denoted as `SetOfPreferred Str` or `SetOfPreferred Int` type. Recall that the SetOfPreferred type is used to capture a set of preferred values. In our example, we have used `SetOfPreferred`

Algorithm 1: PRAWN($\mathbf{S_p}$, D_1, D_2, Ψ)

1 **Input**: Two instances D_1 and D_2 of a PRAWN schema $\mathbf{S_p}$, and a preference specification Ψ.
2 **Output**: $D_1 \uplus_\Psi D_2$.
3 **case** *D_1 and D_2 are SetOf types*
4 Let D_1' (resp. D_2') be the set of all elements in D_1 (resp. D_2) whose keys do not occur in D_2 (resp. D_1);
5 Let $T' = \emptyset$;
6 **for** *every $t_1 \in D_1$ and $t_2 \in D_2$ such that t_1 and t_2 have the same key value* **do**
7 $T' = T' \cup$ PRAWN($\mathbf{S_p'}$, t_1, t_2, Ψ), where $\mathbf{S_p'}$ denotes the type of t_1;
8 Let *Res* = $D_1' \cup D_2' \cup T'$;
9 **return** ■ *Res*, where ■ represents all time points in the temporal contexts of D_1 and D_2;
10 **case** *D_1 and D_2 are both Rcd types*
11 Let D_1 be Rcd[$p_1 : u_1, \ldots, p_m : u_m, \ldots$];
12 Let D_2 be Rcd[$q_1 : v_1, \ldots, q_n : v_n, \ldots$];
13 Wlog, assume $p_i = q_i$, where $1 \leq i \leq k$, and $k \leq \min(m, n)$;
14 Let w_i = PRAWN($\mathbf{S_1}$,u_i,v_i,Ψ), where $1 \leq i \leq k$, and $\mathbf{S_1}$, ..., $\mathbf{S_k}$ denote the schemas of u_1, ..., u_k respectively.
15 Let *Res* = (Rcd[p_1:w_1,. . .,p_k:w_k,
16 p_{k+1}:u_{k+1}, . . ., p_m:u_m,
17 q_{k+1}:v_{k+1},. . .,q_n:v_n, ...]);
18 **return** Res;
19 **case** *D_1 and D_2 are SetOfPreferred[τ] types, where τ is Str or Int*
20 **for** *every* $c_1(d_1) \in D_1$ **do**
21 **for** *every* $c_2(d_2) \in D_2$ **do**
22 Let ($c_1'(d_1)$,$c_2'(d_2)$) = resolve($c_1(d_1)$, $c_2(d_2)$, Ψ);
23 Replace $c_1(d_1) \in D_1$ with $c_1'(d_1)$;
24 Replace $c_2(d_2) \in D_2$ with $c_2'(d_2)$;
25 **return** $c(D_1 \cup D_2)$, where c is a temporal context that represents all time points of the temporal contexts associated to D_1 and D_2;
26 **return** Error; // D_1 and D_2 have different types

Algorithm 2: resolve(d_1, d_2, Ψ)

1 **Input**: Two atomic values d_1 and d_2 of the same type, their associated temporal contexts c_1 and c_2, a set Ψ of preference specifications.
2 **Output**: { $c_1'\, d_1$, $c_2'\, d_2$ }, which is { $c_1\, d_1$ } $\uplus_\Psi$ { $c_2\, d_2$ }.
3 **for** *each* $r_1 \in c_1$ **do**
4 **for** *each* $r_2 \in c_2$ **do**
5 **if** *r_1 and r_2 overlaps in $\mathcal{T}$* **then**
6 **if** *d_2 is preferred over d_1 according to Ψ* **then**
7 // prefer d_2 over d_1
8 replace r_1 with $r_1 \ominus_\mathcal{T} r_2$;
9 **if** *d_1 is preferred over d_2 according to Ψ* **then**
10 replace r_2 with $r_2 \ominus_\mathcal{T} r_1$;
11 **return** { $c_1\, d_1$, $c_2\, d_2$ };

`Int` to capture the quantity of a share.

If D_1 and D_2 are both `SetOf` types, which represent relations, then by the requirements of a PRAWN schema, the elements in each set must be identifiable through keys (see line 3). An element of D_1 is recursively merged with an element of D_2 with the same key. The temporal context of the result consists of all time points of the temporal contexts associated with D_1 and D_2.

The case of record types (`Rcd`), which represent tuples of relations, works similarly to `SetOf` types (see line 10). Values of common attributes are merged via recursive calls, while values of attributes that occur exclusively in the first or second record are simply returned (see fields $k+1$ to m and $k+1$ to n respectively. Similar to the case of `SetOf` type, the temporal context of the result consists of all time points of the temporal contexts associated with D_1 and D_2.

If D_1 and D_2 are both `SetOfPreferred` types, which represent sets of possibly conflicting values, then the temporal contexts of D_1 and D_2 are adjusted through the subroutine `resolve` whenever a conflict occurs. A temporal context is implemented as a set of records. Lines 3-11 of Algorithm 2 first checks whether c_1 and c_2 *overlaps* at some time point by checking whether there is a pair of records in c_1 and, respectively, c_2 that overlaps. The notation $\mathcal{T}$ denotes the set of all time dimensions (e.g., 'asof' and 'knownsince'). We shall exemplify in Section 3.4 that different subsets of time dimensions can be used in general. Formally, we say that two temporal contexts *overlap in time dimensions* $\mathcal{T}$, if they share at least one common time point in the time dimensions given by $\mathcal{T}$. The difference is defined accordingly. The *difference between two temporal contexts w.r.t.* $\mathcal{T}$, denoted as $c_1 \ominus_\mathcal{T} c_2$, is the set of all time points in c_1 that do not overlap with c_2 in the time dimensions given by $\mathcal{T}$. Hence, if the temporal contexts overlap, and d_2 is preferred over d_1 (see line 6), then the temporal context of d_1 is adjusted by removing the time points of d_1 that causes the overlap in the time dimensions given by $\mathcal{T}$. A symmetric case happens in line 9.

We now show that our implementation for PRAWN, described earlier, is correct in that it indeed computes $D_1 \uplus_\Psi D_2$. The proof of Theorem 3.3 (see below) uses induction on the structure of $\mathbf{S}$ and can be found in [2]. Since it makes one pass through D_1 and D_2, it runs in polynomial time in the size of its input.

THEOREM 3.3. *Let D_1 and D_2 be two instances of the same* PRAWN *schema $\mathbf{S}_p$ that satisfy a set Ψ of preference rules. Then,* PRAWN*($\mathbf{S_p}$, D_1, D_2, Ψ) returns $D_1 \uplus_\Psi D_2$. Furthermore,* PRAWN*($\mathbf{S_p}$, D_1, D_2, Ψ) executes in polynomial time in in the size of the input.*

To the best of our knowledge, this is the first algorithm for merging two temporal instances according to a preference specification and the presented algorithm is optimal in the sense that it makes only one pass through both instances.

3.4 Versatility of PRAWN

Next, we demonstrate how PRAWN can capture the integration semantics of several types of real-world integration scenarios, by simply customizing the preference specification and/or the time dimensions used in the computing the difference between two temporal contexts. We discuss three specific applications below and omit the discussion of how the semantics of integrating data (without time) can be achieved by PRAWN which can be found in [2].

Semantics of archiving (i.e., integrating versions of data) The nested merge operator of [8] which is used to archive versions of data can be reduced to PRAWN as follows.

We define the temporal context to contain a single time dimension, say "version". The temporal context of every element in the ith version will contain the interval $[i - \mathsf{now})$ and the temporal context of every element in a later version, say $(i+1)$th version, will contain the interval $[i + 1 - \mathsf{now})$. We define $\mathcal{T} = \{\text{version}\}$ and we define a preference rule

for r_1 and r_2 in $\langle path \rangle$
prefer r_1 if r_1.start(version) $>$ r_2.start(version)

to always prefer values from a later version. Hence, if the value of an element is r_1 in the ith version but r_2 in the $(i+1)$th version, the difference operation will modify the temporal context of r_1 to $[i, i+1)$, and the temporal context of r_2 will continue to have the time interval $[i + 1 - \mathsf{now})$.

Semantics of updates in bi-temporal databases In bi-temporal databases (see [23] for a comprehensive overview of concepts and

related work in this area), there are two notions of time, namely *valid-time* and *transaction-time*. Valid-time denotes the time at which a tuple is valid in the real-world, while transaction-time denotes the time updates are entered into the database. Hence, the transaction times of updates can only increase with each update.

An update on a relation in bi-temporal databases is reduced as input to PRAWN as follows. We model a relation as a SetOf records, where there is a temporal context, with valid and transaction time dimensions, associated with each record. An update is a record with an associated temporal context whose transaction time corresponds to the time the update was sent to the bi-temporal database.

For example, when an update occurs with a valid time interval $[v_3 - v_4)$ at transaction time t, the corresponding temporal context of the update record is c_2 = [valid: $[v_3$-$v_4)$, trans:[t-now)]. Now suppose a tuple occurs in the relation with valid time interval $[v_1$-$v_2)$ and transaction time interval $[t_1$-$t_2)$. The associated temporal context is c_1 = [valid:$[v_1$-$v_2)$, trans:$[t_1$-$t_2)$]. When an update is applied at time t, where $t > t_1$, a conflict occurs if there is a time point $\langle v_5, t_3 \rangle$ that belongs to both c_1 and c_2. In a bi-temporal database, a new tuple is inserted into the relation with the valid and transaction time intervals as described in c_2. At the same time, the old tuple is updated to be associated with the time interval(s) that represent $(c_1 \ominus_{\mathcal{T}} c_2)$, where $\mathcal{T}$ = {valid, transaction}. Overall, two additional copies of the old tuple may need to be inserted to the relation, since as many as three pairs of valid and transaction time intervals may be needed to syntactically represent $(c_1 \ominus_{\mathcal{T}} c_2)$.

The PRAWN preference rule that achieves the above effect is:

for r_1 and r_2 in $\langle relation \rangle$
prefer r_1 if r_1.start(transaction_time) > r_2.start(transaction_time)

In other words, PRAWN prefers values with a later transaction time and the *difference* of two temporal contexts would capture the difference in the set of time points in both time dimensions as is intended in bi-temporal databases.

Semantics of integrating SEC data Like bi-temporal databases, there are two time dimensions ('knownsince' and 'asof') for SEC data. Unlike bi-temporal databases, however, a conflict occurs when there is overlap only in the 'asof' time dimension. In other words, a conflict occurs if two filings of the same share of the same person overlap on an 'asof' time instant. Furthermore, when a conflict occurs, preference is given to the quantity of shares with a later starting 'asof' time.

To capture the semantics of integrating SEC filings with PRAWN, we define the preference specification as ψ_1 in Section 3 and define $\mathcal{T}$ = {asof}. In other words, suppose v_1 and v_2 are two values of the same share that both occur on an 'asof' time instant a, then preference is given to the value (among v_1 and v_2) whose starting 'asof' time is the larger of the two.

To exemplify further, consider the first three SEC filings on the left in Figure 1. After the first two reports are integrated under ψ_1, where $\mathcal{T}$ = { asof }, we obtain:

ticker	shares	
OLP	knownsince: [Jul01-now), asof: [Jul01-Aug23)	300000
	knownsince: [Aug24-now), asof: [Aug23-now)	141

Observe there are no conflicts since the 'asof' time interval of the quantity 300000 has been modified to [Jul01-Aug23). After the third report is integrated, we obtain:

ticker	shares	
OLP	knownsince: [Jul01-now), asof: [Jul01-Aug23)	300000
	knownsince: [Aug24-now), asof: [Aug23-Aug25)	141
	knownsince: [Aug26-now), asof: [Aug25-now)	13415

3.5 Discussions

We discuss several fine points of our implementation next.

Prawn in one pass As long as the elements of every nested set are accessible based on key values (e.g. either by sort order or an index), preference-aware union can be performed in one pass through each instance; A depth-first traversal of the input instances is performed in synchrony followed by a write of the combined content back to disk. The same idea of sorting before merge has been used in [8].

Inheritance and containment In its full generality, a temporal context can be associated with a set, or record, or attribute, in addition to atomic types. Similar to the idea of inheritance of timestamps [8], PRAWN does not store the temporal context c_2 of an element e_2 if e_2 is nested under an element e_1 whose temporal context contains an identical set of time points to the set of time points represented by c_2. For archiving data, [8] has shown that this simple idea can result in substantial space savings for data that does not change frequently. In Section 4, we verify experimentally that we continue to obtain substantial space savings on real data sets even when the temporal contexts that may be composed of multiple time dimensions and the data may involve relatively frequent updates.

Extensions to the algorithm We have presented PRAWN in Section 3.3 on the assumption that conflicts can only occur at the level of atomic types. Hence, if we have two records with identical keys k that occur at the same time, such as R(key:k, A:a_1, B:b_1) and R(key:k, A:a_2, B:b_2), PRAWN will produce R(key:k, A:$\{a_1, a_2\}$, B:$\{b_1, b_2\}$) when no preferences are specified. In the result, the association between the A and B values in the original inputs are lost. In particular, one valid interpretation of the result is the record R(key:k, A:a_1, B:b_2), which was not among the original inputs to PRAWN. However, note that this record is a perfectly valid interpretation given that a_1 and a_2 (resp. b_1 and b_2) are essentially equivalent under the key k, since no preferences are given. One can also extend the algorithm to manage conflicts for complex types (i.e., records or sets). This can be achieved by allowing SetOfPreferred[τ] types, where τ need not be an atomic type and elements under τ need not be keyed. The details are omitted.

4. EXPERIENCE WITH REAL DATA

We implemented PRAWN in Java 6 running on a Xeon Intel 3.4GHz dual core Linux workstation with hyper-threading enabled and 4GB RAM. Our implementation is built on top of XArch [28], and leverages their schema and key specification framework. We experimented with several data sets to analyze the algorithm with respect to compactness, performance, and parallelization.

SWISS-PROT The SWISS-PROT data set, which was used in [26], is a large, regularly updated body of protein sequence data published over many years by different organizations [22]. It encompasses two distinct types of data: hand-curated and machine-generated. As a result, integrating multiple releases requires merging data that conforms to different schemas, and each release includes both new elements and updates to existing elements. We obtained 10 different releases of SWISS-PROT data. Release 40 has over 17 million elements and its file size is 403MB, while release 49 has over 51

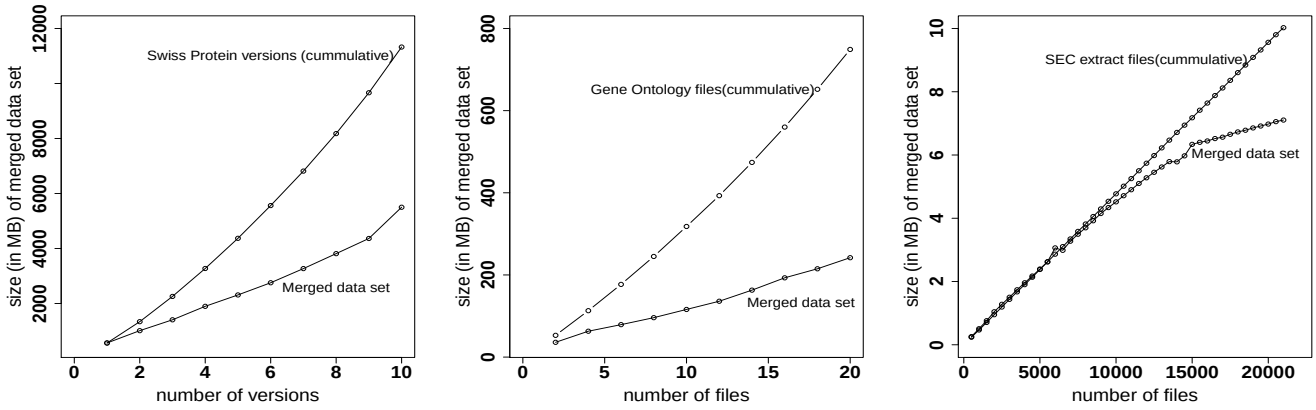

Figure 4: Size of the integrated archives of (a) SWISS-PROT, (b) Gene Ontology and (c) SEC data.

million elements with a file size of 1225MB. As described in Section 3.4, temporal context consists of the version release of SWISS-PROT, and the reported time given by the date of the press release announcing the version was available. Preference is given to information from a later version.

Gene Ontology The Gene Ontology database [11] is a medium size data set in RDF-XML format that contains a hierarchical description of gene characteristics for multiple species. The database is updated daily, and monthly extracts of the ontology going back to 2002 are available via ftp. We chose 20 release files, ranging in size from 27 MB to 53 MB. The temporal context consists of two time dimensions 'since' and 'reported', which we define as the first day of the month in which the database was released, and, respectively, the timestamp of the database. If overlap occurs on the 'since' time, we prefer information with a later since time.

SEC The SEC requires that corporations regularly report information disclosing the stock transactions of its officers and directors [15]. Each report is small, and includes the date the transaction occurred (asof time) and filed (knownsince time) with the SEC. For our experiments, we arbitrarily extracted reports for the second half of 2010. The temporal context is as described in Section 3.4, consisting of both time dimensions. However, overlap is defined only for 'asof' time.

Compactness We show experimentally that the result of preference-aware union is compact. Even though the temporal context takes up some space in general, and can in fact become quite lengthy for elements that undergo frequent changes, the total storage required for representing the integrated archive and temporal context remains small compared to storing individual versions or reports. The graphs in Figure 4 show more detailed analysis for these three datasets. For each data set, the graph shows the number of the files to be merged on the x-axis and the total size of the file, as produced by preference-aware union on these files, on the y-axis. As shown, the changes in the consecutive sizes of the merged data set are substantially less than the cumulative size of each data set. For example, a merged file that contains all 10 releases of SWISS-PROT is less than 50% of the cumulative size of the releases themselves, and a merged file that contains all 20 versions of the Gene Ontology database is less than 30% of the cumulative size of the version files. The storage savings is because a change to an element only adds the new value and modifies the temporal context for previous values, results that are similar to those observed in [8]. Figure 4(c) illustrates a different storage pattern for the SEC data, with the merged data set size after 20,000 files to be about 30% less than the cumulative file size. This is because each filing is relatively small, and the temporal context makes up a larger percentage of the data itself.

We chose a fairly verbose representation for temporal context for readability purposes, and we chose to store it uncoalesced [6]. Thus, each time an entity is seen in the merge, its temporal context is updated. As a result, the context itself can become quite lengthy for large data sets in which entities are repeated across updates, even if they are unchanged. This is true for both SWISS-PROT and Gene Ontology since each release represents a complete version of the data, most of which are unchanged. For the SWISS-PROT data, temporal context annotations make up 50% of the total file size of the final archive, and for the Gene Ontology data, the context makes up almost 60%. The SEC data, on the other hand, had a much higher percentage of unique entities, and the temporal annotations make up only 13% of the size of the final archive.

Scalability We applied preference-aware union to merge successive versions of the SWISS-PROT data, and successive versions of the Gene Ontology data. For both of these cases, the input to the preference-aware union algorithm is the data set containing the cumulative merged versions, and the new version. The graphs in Figure 5(a) and Figure 5(b) show the time in seconds (y-axis) it takes to merge the SWISS-PROT and Gene Ontology data with respect to the file size in MB of each release (x-axis), and the dashed line shows a linear fit of the data points for comparison. Sorting was done according to [8] and [26] and the time shown does not include the time to sort the new file to be merged. As shown in the graphs, the results are consistent with those reported in [8] and the execution time is roughly proportional to the size of the instances to be merged. This is because the algorithm reads both input instances once (in this case, the cumulative merged file and the new file to be merged), and writes their merged content back to disk. For a given execution of the algorithm, let m_1 and m_2 represent the sizes of the two files to be merged. In the worst case, $m_1 = m_2$, both files contain a distinct set of entities, and algorithm runs in $m+m+2m$, or $O(m)$ time, where $m = m_1$ (or m_2).

Parallel implementation of preference-aware union The serial implementation of preference-aware union can easily be applied to data sets with a small number of instances, since each instance can be merged in a single pass. The SEC data, however, is made up of over 20,000 instances. Therefore, a more efficient implementation of preference-aware union is to build up a larger result by repeatedly unioning individual files in parallel. The algebraic properties described in Section 3.2 imply that it is possible to parallelize the algorithm in this way and still guarantee a unique result (modulo representation of time).

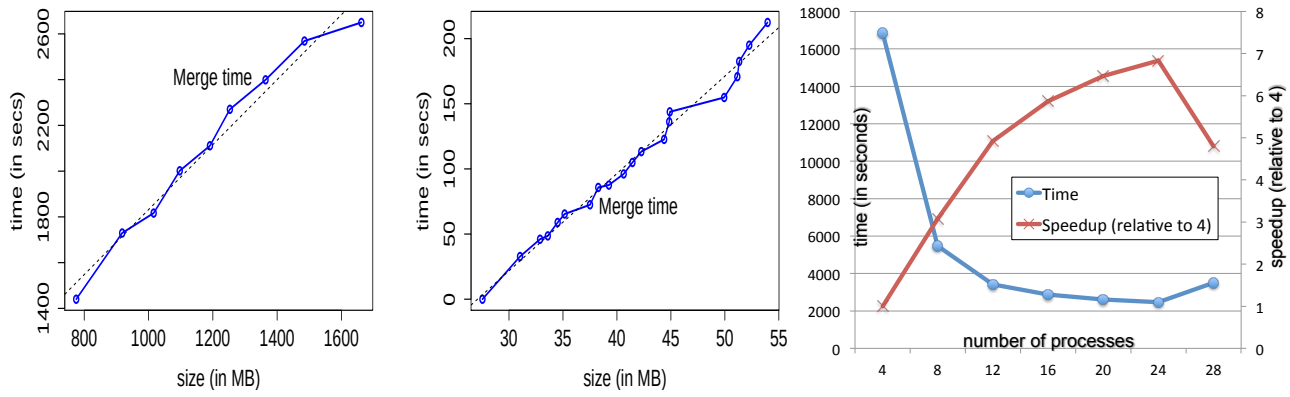

Figure 5: (a), (b): Time to PRAWN SWISS-PROT and Gene Ontology data. (c) time and speedup to PRAWN SEC data in parallel.

We tested the parallel version of PRAWN on the SEC data with a reduction factor r=2, and the results are shown in Figure 5(c). The graphs show consistent speedup and resource utilization, with the largest speedup occurring with 24 processes, indicating resource utilization of the test machine was at its maximum. It should be noted that in each iteration of the parallel version, each parallel process essentially runs the serial version of the algorithm, which requires reading both input files once, and writing out the merged file, once for each file assigned to that process. This is because the merged data is stored as one contiguous file. These results provide evidence of the performance gains for PRAWN with a partitioned file organization and parallel execution framework, which we briefly discuss below and call out as an area of future work.

Experience with Large Scale Datasets We tested the scalability of PRAWN with even larger datasets by using a Twitter social media stream. The base data originated in the GNIP Decahose Twitter Stream [1] and contains a random sample of 10% of the entire public content posted on Twitter in the first three weeks of August 2012. The total size of the base data is 2.25TB, containing approximately 750 million tweets from 50 million users worldwide. The base data of individual tweets was passed through a social media analytics flow [21] expressed in the HIL language [20]. The high-level specification is deployed as a sequence of MapReduce jobs. For our experiments with Twitter data, we used a Hadoop cluster consisting of 10 nodes, each with 8 cores and 32GB RAM. The flow first employed text extraction techniques on the text and metadata associated with each input tweet and inferred attributes such as name, gender, home location, and occupation for the author of the tweet, as gathered from each single input tweet in isolation. The resulting dataset was about 800 GB in size. The next and final phase of the flow created individual user profiles by aggregating all the data available for each user. Our PRAWN operator was employed as part of this aggregation phase, and was integrated as a user function in the analytics flow. The attribute values inferred for gender, home location and occupation were associated with a unidimensional temporal context, bounded by the time when the tweet was authored, and by the present time. [2] Later tweets were preferred over earlier tweets for conflict resolution.

We first examined the scalability of the deployment of PRAWN over MapReduce using a subset of the Twitter data, covering the first week of August 2012. Details are omitted for brevity, but results show consistent speedup as the number of nodes increases. For remaining experiments, we used all 10 nodes and considered Twitter datasets of increasing size, ranging from one day to three weeks. We computed an overall measure of attribute conflicts in the datasets. We defined this as the sum, over all users and all attributes of interest, of the number of values that had to be resolved by PRAWN.

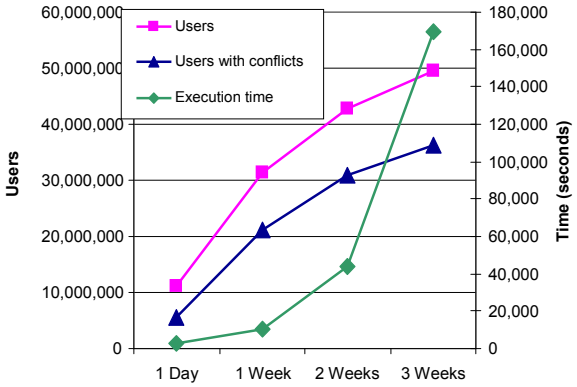

Figure 6: Total num. of users, num. of users for which conflict resolution was performed by PRAWN, and execution time.

The execution time required for each of the datasets of increasing size was measured and is presented in Figure 6. The execution time increases faster than the number of conflicts in the dataset. This is expected, since each invocation of PRAWN, in this instantiation, is essentially a quadratic process in the size of sets of attribute values that need to be resolved across time. While PRAWN takes advantage of the MapReduce infrastructure for its distributed file organization and to leverage more resources when available, individual invocations of PRAWN are not parallelized deeper. As future work, we plan to further parallelize the computation inside each call to the PRAWN operator.

5. TEMPORAL QUERIES OVER THE INTEGRATED ENTITIES

The result of applying preference-aware union to one or more instances is an integrated repository that contains a concise and complete temporal history of all entities. The result instance, including the temporal context, is human readable, making it easy to visually explore the history of an entity, and machine readable, making it possible to explore the history of an entity by standard languages

[1] gnip.com/sources/twitter

[2] Note that we chose a single dimension of time for simplicity of the discussion; it is straightforward to add additional time dimensions as was done with the other data sets used for experiments.

for hierarchical data, such as XPath for XML, and Jaql, an language for manipulating JSON data. While the focus of the current work has been on the preference-aware union operator, we show that it is straightforward to support two important subclasses of temporal queries: *pure timeslice* queries and *range timeslice* queries [27]. We will use Jaql to demonstrate pure timeslice and range timeslice queries over an instance produced by PRAWN. Suppose the variable `tweeters` was bound to the output of the PRAWN operator, and the schema for an entity in the output included attributes called `userid` (which is also the key), `gender`, `location`, and `occupation`, and suppose that `gender`, `location`, `occupation` were all annotated with temporal context. The following pure timeslice query (in which the timeslice is over all time) returns all occupations across time for a given tweeter:

```
tweeters -> filter ($.userid == "80376002") ->
   transform($.occupation) -> expand;
```

To support timeslice queries for specific periods of time, we have implemented a Jaql function `inTemporalContext(entity, timeVector)` that operates on JSON objects annotated with temporal context. The function takes takes as input entity represented as a JSON object and a vector of time intervals and returns true if the temporal context of the object contains the time interval. This function can be used to execute a pure timeslice query at a specific point or interval of time, e.g., to find all locations of a tweeter on a given day:

```
tweeters -> filter ($.userid == "80376002") ->
 filter (inTemporalContext($,"2012-08-08")
   transform($.location) -> expand;
```

This function can also be combined with other Jaql operators to support range timeslice queries. For example, the following Jaql expression will return all tweeters whose occupation was 'teacher' at a specific point in time (or interval of time):

```
tweeters -> filter ($.occupation ->
 filter (inTemporalContext($,"2012-08-08")
   and ($.value == "teacher"))) != [];
```

6. RELATED WORK

Conflict resolution operators, preferences Data fusion techniques for combining (conflicting) information from different sources without time have been extensively studied. (See, for instance, the survey [5] and tutorials [13, 14].) Our PRAWN operator is close in spirit to the *match-join* operator [35] and the *merge* and *prioritized merge* operators [19]. When two records, such as R(key:1, A:2, B:3, C:4) and R(key:1, A:5, B:6, D:7), are match-joined, a total of four records will be created: R(key:1, A:2, B:3, C:4, D:7), R(key:1, A:2, B:6, C:4, D:7), R(key:1, A:5, B:3, C:4, D:7), and R(key:1, A:5, B:6, C:4, D:7). On the other hand, under a suitable PRAWN schema, PRAWN will derive a compact representation R(key:1, A:{2,5}, B:{3,6}, C:4, D:7), where the same set of four records can be derived from the product of the nested set of elements. With suitable extensions to the PRAWN algorithm, PRAWN can also derive the same output obtained by the merge operator, which consists of two records: R(key:1, A:2, B:3, C:4, D:7) and R(key:1, A:5, B:6, C:4, D:7). Prioritized merge derives R(key:1, A:2, B:3, C:4, D:7) by preferring tuples from the left relation. By specifying our preference rule to always prefer values from the left relation, PRAWN obtains the same result.

To the best of our knowledge, PRAWN goes beyond prior work on conflict resolution operators as it can manage and resolve conflicts across time through declarative use of preference rules.

A *prioritized repair* [32] is an extension of the concept of a repair in inconsistent relational databases which accounts for priorities among conflicting tuples. Unlike our preference rules, the work of [32] assumes that an explicit priority relation, which defines preferences between tuples, is available. They studied various notions of optimality of instances with respect to a given priority relation under the class of *denial constraints*, which includes the class of key constraints as a special case. They gave algorithms that, given an inconsistent relation, a priority relation, and a set of denial constraints, will generate *one* optimal repair (in their sense). In contrast, our polynomial-delay algorithm enumerates *all* optimal repairs at a given time point for hierarchical temporal data under key constraints.

In [16], a language for specifying priorities among spanners in information extraction systems has been proposed. However, unlike [16], our preference rules are specified over temporal data that may be hierarchical. Recently, [18] also leveraged preferences in cleaning data in the process of mapping transformations. Trust policies [33], which is a type of preference, have also been used to understand how one can prioritize updates.

PRAWN distinguishes itself from prior work on preferences and trust policies by accounting for time in a hierarchical data model.

Bitemporal databases There has been solid foundation of work in relational bitemporal databases [10, 23], and in Section 3.4, we have shown that PRAWN can be used to capture valid-transaction time semantics of bitemporal databases. However, the converse is not true in general as bitemporal databases only enable applications to manipulate valid time, and assume that updates are received in order of increasing transaction time. Specifically, bitemporal databases may return the wrong integrated result using its valid-transaction time semantics for managing SEC data.

To exemplify, consider the 4th and 5th SEC filings on the left of Figure 1, which were clearly reported out of order. Suppose these reports are updates to a bitemporal database with transaction times given by t_1 and t_2 respectively. After an update for the 4th filing, a bitemporal database would record that 0 shares of BRT were held from Jul14-now with a transaction time of [t_1-now). After a database update for the 5th filing, the bitemporal database would record that 0 shares of BRT were held from [Jul14-now) with a transaction time of [t_1-t_2), and 1820 shares were held from [Jul09-now) with a transaction time of [t_2-now). If the updates were processed in the opposite order (i.e., the filing with 1820 BRT shares before 0 BRT shares), then after both updates are processed, the bitemporal database would record that 1820 shares were held from [Jul09-Jul14) with a transaction time of [t_1-now) and [Jul14-now) with a transaction time of [$t_1 - t_2$). In addition, 0 BRT shares were held from [Jul14-now) with a transaction time of [t_2-now). Thus, the number of shares held on Jul14 is 0 or 1820 depending on the order in which the updates were applied. A more detailed description of the difference can be found in [2, 31].

Others Prior work aimed at corroborating the truth from conflicting information from different sources (e.g., [12, 17, 30, 36]) constitute another line of related work. In these systems, truth is determined by majority voting schemes, probabilistic models (over time), and/or trust on data sources. In contrast, PRAWN is a deterministic truth finding framework over time based on user-specified preference rules. As we have described towards the end of Section 1, the general integration workflow is likely to deploy both rule-based and probabilistic or machine-learning based approaches.

Complex event processing and data streams systems make decisions based on continuously streaming data that may arrive in order or out-of-order and for which the time element associated with data values may be known with certainty or may be imprecise (e.g., [4, 37]). These systems, however, do not resolve violations through user-specified preferences as part of the integration process to produce a consistent integrated result. Finally, three types of temporal

semantic heterogeneity in data integration were described in [38]. The focus of [38] was to develop a system to cope with the heterogeneity of objects over time, differences in the semantics of the same attribute over time, and differences in time representation. It does not resolve conflicts across time based on preferences.

7. CONCLUSION AND FUTURE WORK

Building a complete and consistent profile of an entity from multiple temporal data sources requires time-specific knowledge to be carefully maintained as new information is integrated. We have developed a simple and yet versatile framework with a corresponding implementation for integrating temporal data that may contain different dimensions of time. Our framework integrates temporal data and resolves conflicts based on user-defined preference rules. The consistent interpretations of the remaining conflicts at a time point, if any, can be systematically enumerated. We have shown that the operator is particularly suitable for data integration and parallelization. At the same time, it is also general and can be easily specialized to capture different integration semantics of several real-world applications across time.

Our experience with real data confirm that our technique can be effectively used to integrate temporal data. Finally, since our integrated archive is immediately representable in XML or JSON, we show how some useful temporal queries about entities can be easily answered with the archive.

This work is a step towards several directions of research (theory and systems). We plan to explore several extensions to our operator. E.g., allow finer-grained specification of preferences, particularly when preferences may be qualified by time. In a separate direction, we plan to explore how concepts in consistent query answering [3] can be carried over inconsistent temporal databases. Finally, we also have plans explore multi-dimensional access methods and indexing methods to further improve PRAWN's performance.

Acknowledgements The authors would like to thank Laura Haas, Lucian Popa, and the VLDB reviewers for their valuable feedback. This research is supported in part by NSF grant IIS-1450560, Google, and IBM Faculty Awards.

8. REFERENCES

[1] B. Alexe, D. Burdick, M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, I. Stanoi, and R. Wisnesky. High-level rules for integration and analysis of data: New challenges. In *In Search of Elegance in the Theory and Practice of Computation*, pages 36–55, 2013.

[2] B. Alexe, M. Roth, and W.-C. Tan. Preference-aware integration of temporal data. Technical report, UC Santa Cruz, April 2014. `www.soe.ucsc.edu/research/technical-reports/UCSC-SOE-14-04`.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[4] R. S. Barga, J. Goldstein, M. Ali, and M. Hong. Consistent streaming through time: A vision for event stream processing. In *CIDR*, pages 363–374, 2007.

[5] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2009.

[6] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in temporal databases. In *VLDB*, pages 180–191, 1996.

[7] P. Buneman, A. Deutsch, and W.-C. Tan. A deterministic model for semistructured data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1998.

[8] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving scientific data. *ACM TODS*, 29:2–42, 2004.

[9] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*, pages 827–832, 2013.

[10] J. Chomicki and D. Toman. Temporal databases. In *Foundations of Artificial Intelligence*, pages 429–467. Elsevier, 2005.

[11] G. O. Database. Gene Ontology Database. `http://www.geneontology.org/`.

[12] X. L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1):562–573, 2009.

[13] X. L. Dong and F. Naumann. Data fusion - resolving data conflicts for integration. *PVLDB*, 2(2):1654–1655, 2009.

[14] X. L. Dong and D. Srivastava. Big data integration. *PVLDB*, 6(11):1188–1189, 2013.

[15] The EDGAR Public Dissemination Service. `http://www.sec.gov/edgar.shtml`.

[16] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *ACM PODS*, pages 164–175, 2014.

[17] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *WSDM*, pages 131–140, 2010.

[18] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and cleaning. In *ICDE*, pages 232–243, 2014.

[19] S. Greco, L. Pontieri, and E. Zumpano. Integrating and managing conflicting data. In *Ershov Memorial Conf.*, pages 349–362, 2001.

[20] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: A High-level Scripting Language for Entity Integration. In *EDBT*, pages 549–560, 2013.

[21] M. A. Hernández, K. Hildrum, P. Jain, R. Wagle, B. Alexe, R. Krishnamurthy, I. R. Stanoi, and C. Venkatramani. Constructing consumer profiles from social media data. In *BigData Conference*, pages 710–716, 2013.

[22] E. B. Institute. Swiss-Prot Protein Knowledgebase. http://www.bi.ac.uk/uniprot/, cited on September 15, 2012.

[23] C. S. Jensen and R. T. Snodgrass, editors. *Temporal Database Entries for the Springer Encyclopedia of Database Systems*. Springer, 2009. `www.cs.arizona.edu/~rts/pubs/TRmerged.pdf`.

[24] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending existing dependency theory to temporal databases. *IEEE TKDE*, 8(4):563–582, 1996.

[25] D. Johnson, C. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Info. Process. Lett.*, 27(3):119–123, 1988.

[26] I. Koltsidas, H. Müller, and S. D. Viglas. Sorting hierarchical data in external memory for archving. *PVLDB*, 1(1):1205–1216, 2008.

[27] A. Kumar, V. J. Tsotras, and C. Faloutsos. Access methods for bi-temporal databases. In *Temporal Databases*, pages 235–254, 1995.

[28] H. Müller, P. Buneman, and I. Koltsidas. Xarch: archiving scientific and reference data. In *ACM SIGMOD*, pages 1295–1298, 2008.

[29] G. Özsoyoglu and R. T. Snodgrass. Temporal and real-time databases: A survey. *IEEE TKDE*, 7(4):513–532, 1995.

[30] A. Pal, V. Rastogi, A. Machanavajjhala, and P. Bohannon. Information integration over time in unreliable and uncertain environments. In *WWW*, pages 789–798, 2012.

[31] M. Roth and W.-C. Tan. Data integration and data exchange: It's really about time. In *CIDR*, 2013.

[32] S. Staworko, J. Chomicki, and J. Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.

[33] N. E. Taylor and Z. G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *ACM SIGMOD*, pages 13–24, 2006.

[34] G. Weikum, N. Ntarmos, M. Spaniol, P. Triantafillou, A. A. Benczúr, S. Kirkpatrick, P. Rigaux, and M. Williamson. Longitudinal analytics on web archive data: It's about time! In *CIDR*, pages 199–202, 2011.

[35] L.-L. Yan and M. T. Özsu. Conflict tolerant queries in aurora. In *CoopIS*, pages 279–290, 1999.

[36] X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. In *KDD*, pages 1048–1052, 2007.

[37] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *PVLDB*, 3(1-2):244–255, 2010.

[38] H. Zhu, S. E. Madnick, and M. D. Siegel. Effective data integration in the presence of temporal semantic conflicts. In *TIME*, pages 109–114, 2004.