ADVANCED DISTRIBUTED DATA INTEGRATION INFRASTRUCTURE AND RESEARCH DATA MANAGEMENT PORTAL

by

Evgeny Karataev

Dipl.-Ing., Tomsk State University of Control Systems and Radioelectronics, 2011

Submitted to the Graduate Faculty of

the School of Information Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2016

UNIVERSITY OF PITTSBURGH

SCHOOL OF INFORMATION SCIENCES

This dissertation was presented

by

Evgeny Karataev

It was defended on

May 6th, 2016

and approved by

Marek Druzdzel, Associate Professor, School of Information Sciences

Konstantinos Pelechrinis, Assistant Professor, School of Information Sciences

Nicholas Nystrom, Director, Strategic Applications, Pittsburgh Supercomputing Center

Dissertation Advisor: Vladimir Zadorozhny, Associate Professor, School of Information Sciences

ADVANCED DISTRIBUTED DATA INTEGRATION INFRASTRUCTURE AND RESEARCH DATA MANAGEMENT PORTAL

Evgeny Karataev, PhD

University of Pittsburgh, 2016

The amount of data available due to the rapid spread of advanced information technology is exploding. At the same time, continued research on data integration systems aims to provide users with uniform data access and efficient data sharing. The ability to share data is particularly important for interdisciplinary research, where a comprehensive picture of the subject requires large amounts of data from disparate data sources from a variety of disciplines. While there are numerous data sets available from various groups worldwide, the existing data sources are principally oriented toward regional comparative efforts rather than global applications. They vary widely both in content and format. Such data sources cannot be easily integrated, and maintained by small groups of developers.

I propose an advanced infrastructure for large-scale data integration based on crowdsourcing. In particular, I propose a novel architecture and algorithms to efficiently store dynamically incoming heterogeneous datasets enabling both data integration and data autonomy. My proposed infrastructure combines machine learning algorithms and human expertise to perform efficient schema alignment and maintain relationships between the datasets. It provides efficient data exploration functionality without requiring users to write complex queries, as well as performs approximate information fusion when exact match does not exist. Finally, I introduce Col*Fusion system that implements the proposed advance data integration infrastructure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

During my PhD studies I have met many people who helped me along the way and I would like to take this opportunity to thank some of them here.

First, and foremost, I would like to thank my advisor Dr. Vladimir Zadorozhny, for giving me a chance to study and work under his supervision. For his patience, understanding and support. For always believing in me even when I had doubts about myself. This dissertation would not be possible without his guidance, encouragement and wisdom. I would like to thank Dr. Marek Druzdzel, for many interesting classes that I enjoyed taking and for letting me have a workplace in his DSL lab. I always felt at home and many people from the lab turned into my friends. I would like to thank Dr. Konstantinos Pelechrinis and Dr. Nicholas Nystrom for taking time to be on my PhD committee, and providing me with valuable feedback. I would like to thank Dr. Nystrom also for his help and support with the Pittsburgh Supercomputer Center resources.

I am grateful to all the people that surrounded me in the School of Information Sciences. The staff were always very helpful in answering any questions I had.

Last, but not least, I would like to thank my family. My parents, Galina and Pavel, my sisters Tanya and Olga, and my wife Feng, for always being there for me, supporting me in whatever I do and wherever it may take me. I dedicate this work to them.

1.0 INTRODUCTION

The amount of data available due to the rapid spread of advanced information technology is exploding. At the same time, continued research on data integration systems aims to provide users with uniform data access and efficient data sharing. The ability to share data is particularly important for interdisciplinary research, where a comprehensive picture of the subject requires large amounts of data from disparate data sources from a variety of disciplines. While there are numerous data sets available from various groups worldwide, the existing data sources are principally oriented toward regional comparative efforts rather than global applications. They vary widely both in content and format. Such data sources cannot be easily integrated and maintained by small groups of developers.

In this dissertation, I address the challenges in developing a large-scale information integration infrastructure that can be utilized as an efficient tool to support a wide range of interdisciplinary research. The solution that I propose is *to engage a large community of researches to share their data, collectively resolve the data heterogeneities, and harmonize their efforts in data reliability assessment and data fusion.* I introduce *Col*Fusion (Collaborative data Fusion)* – an advanced infrastructure for systematic accumulation and utilization of global heterogeneous datasets based on the collective intelligence of research communities. Col*Fusion efficiently distributes the task of data integration among the data contributors and enables continuous growth of *integrated* repository in a Wikipedia-like manner.

Over the last several decades there has been much research on the various components of digital data curation infrastructure. However, there has been little work on collecting all of the components into an integrated end-to-end system. This work constitutes the first attempt to systematically utilize state of the art as well as to develop novel techniques to implement a global-scale high-performance data integration infrastructure based on collective intelligence.

A major part of the research is focused on historical data integration, performed in conjunction with the Collaborative for Historical Information and Analysis (CHIA) (http://www.chia.pitt.edu/), to provide an immediately valuable test-bed for research in crowdsourcing information integration. The impact on historical research will be significant both nationally and internationally, because CHIA currently involves nine different research groups throughout the U.S. and Europe. Historical data integration is an initial test-bed for Col*Fusion. The proposed information integration infrastructure is general enough to apply to any fields involving large bodies of structured and unstructured textual and numerical data.

1.1 MOTIVATING EXAMPLE

Interdisciplinary research requires data produced by different research group and stored in separate datasets. Consider the following hypothetical question: "*Is there a correlation between population and number of disease cases in United States*?". Also consider four datasets that might be collected/produced by several independent researchers:

- D_1(State, Year, Month, Disease, Number of disease cases) – total number of disease cases in USA aggregated by state, year and month; available as a database dump file.

- D_2(State, Abbreviation) – mapping between USA state names and their abbreviations, e.g. (PA, Pennsylvania); available as a CSV file.
- D_3(State, Year, Month, Country, Precipitation) – precipitation amount in some USA states aggregated by year and month; available as an SPSS file.
- D_4(Population, Year, State*) – population in some USA states aggregated by year and state; available as a STATA file.

The datasets are heterogeneous in many ways: file format, schema and values, e.g. even though variable name is State in the last dataset, actual values are abbreviations of states.

Figure 1 shows a number of steps a researcher would need to do to answer the above question. The required data are stored in D_1 and D_4, however they cannot be directly merged based on State and Year variables, because State in D_4 is represented as a state name abbreviation. In general, some of the datasets might not even share any common variables. Meanwhile, they might be related via other datasets. In the above example, the researcher would have to either edit State variable in D_1 or D_4 or merge them via other dataset, e.g. D_1-D_2-D_4.

Most of the existing data repositories focus only on datasets level metadata (variables metadata can be provided in form of codebook, that cannot be processed automatically). Therefore, interoperability between repositories is reduced to only sharing datasets level metadata or searching for datasets based on datasets level metadata with no support for search of specific variables. In addition, when datasets are downloaded, without variable level metadata it can be hard to know what exactly each variable measures, e.g. is it approximate, is it aggregate, what measure units are, etc.

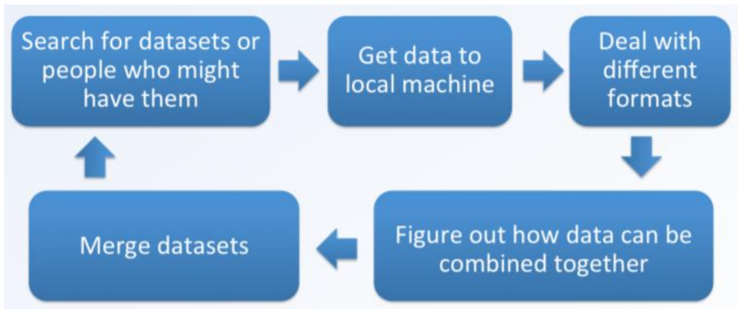

Search for datasets or people who might have them
Get data to local machine
Deal with different formats
Figure out how data can be combined together
Merge datasets

Figure 1: Steps required to answer data-intensive interdisciplinary research questions

1.2 GRAND VISION

We envision a world in which all datasets are publicly available (with appropriate permissions and licensing) and access points to those datasets are not spread over hundreds of different digital libraries and repositories, government and personal websites, etc. The process of data sharing could be as simple as visiting a web site and doing a couple of mouse clicks with no time consuming data preparation and transformation to fit strict format requirements. The data could also be shared automatically from any data manipulation software.

Imagine being credited and recognized for the data you share and seeing your datasets used and/or being evolved over time. The datasets you share are not getting lost among other datasets but instead they are automatically linked with other datasets (even from other disciplines) while preserving data autonomy. The data linkage could bring you new interdisciplinary research questions and new collaboration that you did not realize at the time you created your data.

Instead of searching for locations of useful datasets you just look for the data you are interested in, e.g. the variables that you want to analyze. Such search does not merely results in a list of links to the locations of potentially relevant datasets, but instead the result includes the data items that you are interested in. Moreover, if the data that you need originally located in a number of distinct datasets, the resulting dataset will automatically integrate all those datasets.

Any dataset you look at will also include comprehensive metadata on both datasets level (e.g., title, description, authors, etc.) and variable level (a short description explaining what it stores, the data type, measuring unit, etc.). The dataset will also be provided with provenance information that would include any actions performed on the dataset since it was created. In case of integrated dataset, provenance information would describe how the integration was performed (e.g. which datasets were used, which variables were used, any transformations that were applied, etc.). If the datasets were used in any published work, you would be able to obtain and review a list of relevant papers. You would be able to immediately explore and understand the dataset by looking at data visualizations (graphs, maps, etc.) created and shared by other users. You could also reuse the results of previous research, such as code to analyze the datasets, statistical model, etc.

Imagine if you could join other researchers currently working on datasets that you are interested in and directly communicate with them while working on those datasets. Moreover, you could run complex data analysis algorithms and/or write your own data analysis/visualization program collectively, without the need to download the datasets to your local machine and instead be able to utilize the power of the high-performance cloud computing. After that you could easily share your analysis with the research community. For complex tasks that are out of your competence, you could also hire (for money or other rewards) domain experts, programmers, data analysts, etc.

You could write papers in which all charts are interactive via simple user interfaces and those papers would be published to journals allowing other researchers to try different parameters to better understand your paper. In such papers, the data and analysis code become a major part of the paper and not just something nice to have. This would allow other researchers to validate your studies.

Imagine that all features explained above are implemented in a simple to use web application that requires neither complicated installation nor considerable learning efforts. Below I will elaborate on the approach and infrastructure that implement the above vision.

1.3 OBJECTIVE

In this dissertation, I present my work towards the grand vision that I explained above. I propose novel advanced infrastructure for large-scale data integration based on crowdsourcing techniques. In particular, I propose novel architecture and algorithms that answer the following questions:

- How to store dynamically incoming heterogeneous datasets efficiently to enable both data integration and data autonomy.
- How to combine machine learning algorithms and human expertise to perform efficient schema alignment and maintain relationships between the datasets.
- How to provide efficient data exploration functionality without requiring users to write complex queries.
- How to perform approximate information fusion when exact match does not exist.

I design and develop Col*Fusion system that implements the proposed advance data integration infrastructure as well as other functionality to realize the grand vision.

The proposed architecture is based on the crowdsourcing techniques and users of the Col*Fusion play central role in the goal of creating high quality large scale integrated data repository. Meanwhile, I am not focusing on specific crowdsourcing related issues, such as how to provide incentive for people to contribute their datasets.

1.4 ORGANIZATION OF THE DISSERTATION

The reminder of this dissertation is structured as follows. Chapter 2.0 covers the necessary background information and literature review. In Chapter 3.0 I describe proposed approaches and algorithms to store, virtually integrate and explore heterogeneous datasets. Chapter 4.0 covers implementation details, introduces Col*Fusion system and reports on its real life usage. Chapter 5.0 addresses the problem of fusing datasets during ad-hoc join queries. I summarize my work and conclude with a discussion of possible future work in Chapter 6.0.

2.0 BACKGROUND AND RELATED WORK

In this chapter I describe main concepts that are relevant to the proposed work and that are either used as building blocks of the proposed infrastructure or serve as motivation. For each concept I also provide literature review and try to contrast related work to the proposed work.

2.1 DATA INTEGRATION

The main goal of data integration is to provide a user with unified view of a number of autonomous and heterogeneous data sources [113]. The challenge of data integration has been actively explored for more than 30 years beginning with the Multi-Base System [111]. Resolving data heterogeneities has been the focus of active research and development [18][63][28][79]. Data integration is a complex process consisting of several activities such as schema matching, record linkage, querying and search over integrated sources, as well as keeping track of lineage and provenance. There are numerous tools for efficient mapping of data sources in a homogenous schema with proper data cleaning, standardization of names, conversion of data types, duplicate elimination, etc.

There are many approaches to design and implement data integration system that address some or all of the challenges mentioned above, but broadly speaking they can be split into two groups:

- Top-down approach
    - Data Warehousing (DW)
    - Virtual Integration (VI), also called View-based Data Integration
- Bottom-up approach
    - Peer-to-peer (P2P)
    - Dataspaces and Pay-as-you-go data management
    - Linked Data

2.1.1 Top-down approaches: Data Warehousing and Virtual Integration

A well-established top-down approach to data integration varies from data warehousing to virtual databases architectures [55] and relies on designing a pre-defined global schema. The global schema is the unified view that data integration system exposes to its users and based on which users post queries to the system. Depending on how data integration system handles data sources and execute queries, top-down approach can be divided into two approaches: Data Warehousing and Virtual Integration.

The main limitation of top-down approaches is the global schema. It is problematic to develop and maintain a predefined schema for many data integration scenarios, especially if the data sources are added dynamically. Another limitation of the top-down approach is that it usually requires a centralized management. When data from several domains need to be integrated, the database administrator might need to have domain knowledge from all domains and/or it will be impossible to define a global schema and the other parts of the data integration system.

2.1.1.1 Data Warehousing Approach Traditionally a data warehouse is defined as a tool for decision support. For example, Inmon [92] defines data warehouse as "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making" and Chaudhuri and Dayal [37] define data warehouse as "a collection of decision support technologies, aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions".

Figure 2 shows typical data warehouse architecture (the figure is taken from [37]).

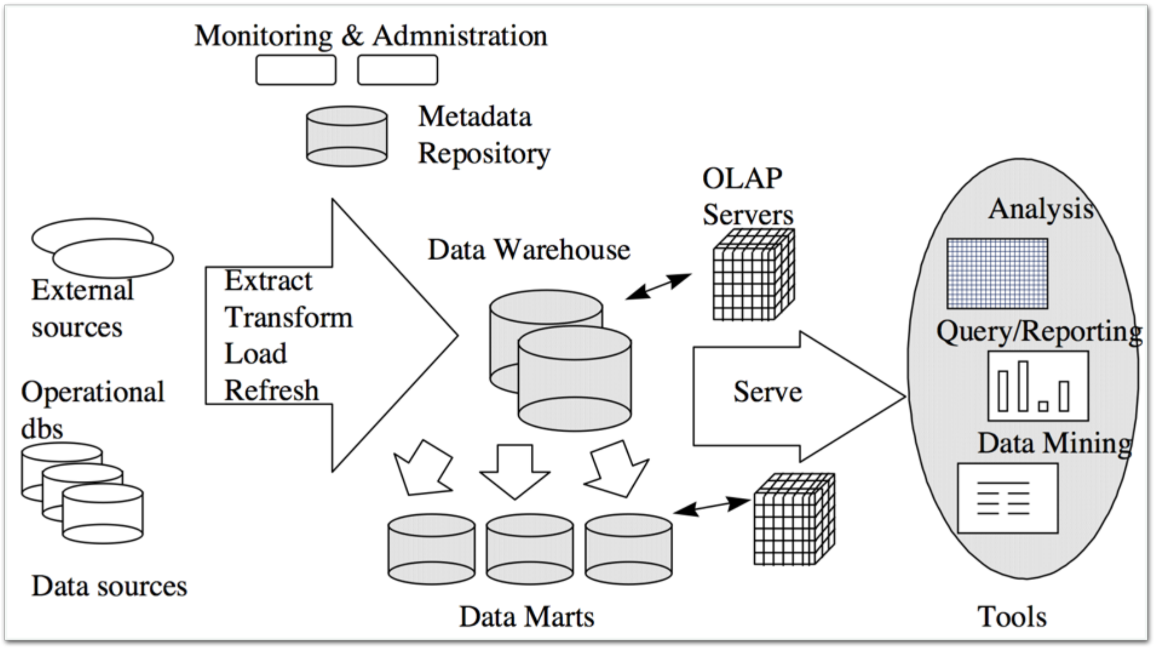


Figure 2: Data Warehousing Architecture

Defining a data warehouse involves two main tasks [55]:

- Performing target database schema and physical design,
- Defining a set of extract/transform/load (ETL) operations.

Data from all the data sources is loaded into the target database periodically (e.g. every night) via the ETL process. Designing the ELT process might not be very straightforward as this is the stage where data source heterogeneity has to be resolved.

DW approach has two main applicability limitations:

- Because the data are loaded into the target database periodically, at a time of the query, DW might have obsolete data.
- Data warehouses usually store aggregated data, therefore some data are lost during transformation process.

On the other hand, DW approach has several advantages:

- Queries over the warehouse don't put any execution load on the data sources since all the data reside in the DW.
- DW supports complex, interactive, exploratory analysis of very large data (OLAP [37])

A data warehouse approach can be easily implemented on top of any of the popular database management systems (e.g. Oracle or Microsoft SQL Server). Examples of data integration in WH in the literature include [33].

The data warehouse approach can be used when all data sources are known and integrated dataset will be used mostly for archival purposes or complex queries for exploratory analysis over [historical] records. It should not be used when integrated data need to be always up to date with all data sources.

2.1.1.2 Virtual Integration Approach Virtual data integration approach is similar to the data warehouse approach in that they both require a global schema. In the virtual data integration approach, however, the global schema is usually called mediated schema and it is "not materialized" (hence it is virtual). "Not materialized" means that data remain in the data sources and accessed at the query time [55].

Simplified virtual integration architecture is presented on Figure 3 (the figure is taken from [55]). When a user posts a query over the mediated schema, the virtual integration system translates the query in terms of data sources and accesses those data sources that have required data. In order to do that the system needs to have enough information about each data source. Thus the key component of the virtual data integration approach is the *source descriptions* that specify the properties of the sources that the system needs to know [115] (Schema mappings are the major part of the source description, see more on Schema mappings in Section 2.1.3).

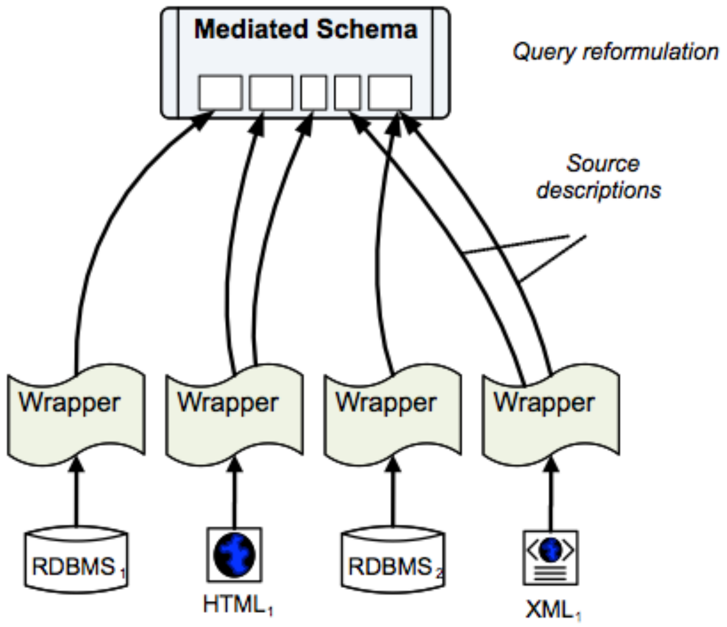


Figure 3: Simplified virtual data integration architecture

To address the source heterogeneity problem, each source is associated with a wrapper that is responsible for communication with that data source. The communication includes posting queries to the data source, receiving answers, and possible applying some basic transformation [55]. A number of works has been done on Web data integration and wrapper/mediator architectures to access heterogeneous Web data sources [194][192][193].

Virtual data integration has several major applicability limitations:

- Complex, data intensive queries cannot be executed efficiently:
    - The data are not loaded into a central repository, but accessed at the query execution time
    - Data transformations happen at the query execution time
    - Complex query plans – harder to optimize
- At any time, any data sources might become unreachable.

Virtual data integration approach has several advantages though:

- Result of a query posted over mediated schema will always have up-to-date data with the data sources because data are pulled from sources during query execution.
- The relationships between mediated schema and data sources are explicitly stated and not hidden inside a particular implementation (e.g. ETL operations).
- Depending on the schema mapping language, new resources can be added relatively easily.

Examples of data integration systems that uses virtual data integration approach include TSIMMIS [72], Garlic [34], Information Manifold [107].

Virtual data integration approach should be used when query results cannot have obsolete data and when queries are not data intensive.

2.1.2 Bottom-up approaches: Peer-to-peer, Pay-as-you-go, and Linked Data

As shown above, constructing a queryable global schema is one of the major challenges in getting a data integration system deployed. The bottom-up approach doesn't require a global schema and, by design, supports seamless integration of new data sources. Below I discuss three types of bottom-up approaches: Peer-to-peer, Pay-as-you-go, and Linked Data.

2.1.2.1 Peer-to-peer approach Peer-to-peer integration systems [1][19][104][82] were inspired by Peer-to-Peer file sharing systems. The main difference from previously reviewed approaches is that they drop requirement for a *single* unified view, allowing queries to be posted over any sources schema. P2P data integration can be understood as a generalization of view based data integration with more than one global schema.

P2P architecture (Figure 4 is taken from [55]) consists of on a set of peers (data sources) and two kinds of mappings: *storage descriptions* and *peer mappings* [55]. Each peer has a schema, called *peer schema*, which is exposed to other peers. Peer schema is a logical schema; the data are stored in a database, called *stored relations*. The storage descriptions specify how to map the data from stored relations to peer schema. In fact, some peers can be complex data management systems (e.g. can be a warehouse) themselves, while some peers might not have any stored relations and act just as a mediated schema.

Peers specify peer mappings that relate their peer schemas. Every peer needs to provide semantic mappings only to a set of neighbors and thus form a network of data sources. More complex integrations emerge as the system follows semantic paths in the network [1]. The transformation, schema mappings, record linkage and all other data integration challenges are address at the peer's level instead of a global level.

Examples of P2P integration systems include BestPeer++ [40], Piazza [81], PeerDB [137].

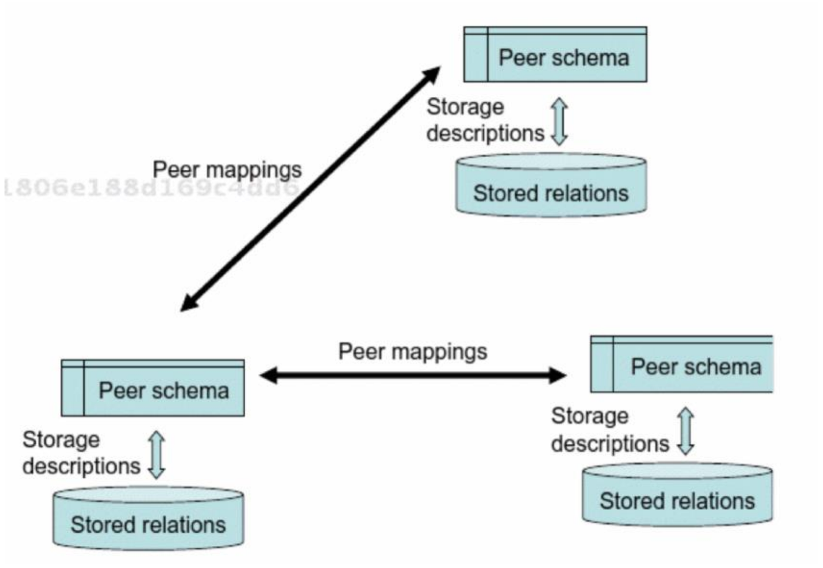


Figure 4: Simplified Peer-to-peer data integration architecture

The limitations and disadvantages of P2P and CDSS systems are similar to virtual data integration approach:

- Because the data are not pulled into one repository, query processing is (a) more completed and (b) slower to execute.
- Because there is no global schema, search need to be performed based on peer's schema. Some work has been done in the area of keyword and top-k search over databases [88, 171, 182]. However, because of the high complexity heuristics are used which might result in not complete data.
- Because there might be several ways to answer a query system need to deal with the problem of data completeness and trust to peers.

2.1.2.2 Pay-as-you-go approach One of the major limitations for all the approaches reviewed so far is long time to setup before any services can be provided. To address the problem, dataspace systems with pay-as-you-go approach were proposed [83][69][125]. The main idea behind this approach is stated by Halevy as "…offer some services immediately without any setup time, and improve the services as more investment is made into creating semantic relationships" [84].

As this approach is relatively new, not many systems were developed yet. Google Fusion tables [73] and OpenRefine [181] systems are examples of pay-as-you-go data integration. They are cloud-based services for data management and integration thus require no installation. Users starts to work with their data without prior configuration and can integrate their data with other data available online on demand.

The disadvantages and limitation of this approach:

- If data integration happens on demand, then the issue of trust in data sources is more challenging here because the system might not have enough information to assess it.
- End user might need to know how to integrate often unfamiliar to him or her resources.

2.1.2.3 Linked Data approach All of the approaches above are focused on data integration on a small scale. For example, an enterprise that have a number of operational databases might have a task to integrate them; or a group or researches want to integrate their datasets; etc. As the result we have isolated clusters of integrated data sources. Much bigger goal is to create a global data space where any data instance can be reached. Semantic Web with Linked Data principles has that goal: "The Semantic Web isn't just about putting data on the web. It is about making links, so that a person or machine can explore the web of data. … With linked data, when you have some of it, you can find other, related, data." [120]

To manage globally distributed data, Semantic Web infrastructure uses Resource

Description Framework (RDF) as a data model [7]. The RDF data model is designed for the integrated representation of information that originates from multiple sources, is heterogeneously structured, and is represented using different schemata [86]. Every thing is called a resource in RDF and represented as a number of triples. Each triple consists of *subject* (refer to the thing itself, e.g. id of the row in the relational model), *predicate* (refer to the name of the property of the thing, e.g. could be thought of as an attribute name of a record in relational model), *object* (refer to the value of the property, e.g. the value of an attribute of a record in the relational model). Two types of RDF triples can be distinguished:

- *Literal Triples* have a simple data type (e.g. string, number, etc.) as the object and they describe properties of the resource.
- *RDF Links* describe the relationships between two resources and can be though of as foreign keys in relational model.

It is convenient to view the triples as a directed graph in which each triple is an edge from its subject to its object, with the predicate as the label on the edge.

In the past few years this approach has been used in publishing government data sets. Examples include www.data.gov, linkeddata.org. However simply following the Linked Data principles to publish research data would not ensure its reusability due to many reasons (data provenance, quality, credit, attribution and reproducibility). For example, publishing data out of context would fail to reflect the research methodology and respect the rights and reputation of the researcher [65].

This approach might look similar to the Virtual Data integration approach with the difference that the "mapping" is on the record level, instead of the schema level. Therefore, the limitations are also similar:

- Complex query might be slow due to the fact the all data are distributed globally.
- At any time, any data source can go offline.

To conclude, I would like to state that each data integration approach has its pros and cons, however all of them require some kind of data administrators either to design a target or mediated schema, or create ETL pipeline or schema mappings, etc. With a number of available data sources and the size of them constantly growing data integration cannot be manual, few humans' effort. Even though automated algorithms based on machine learning techniques are developed, as noted by Halevy they are not foolproof [84]. Therefore, there is a need for a new approach.

2.1.3 Schema Matching and Schema Mapping

An important task in any data integration system (no matter what architecture is used) is to align schemas of the data sources that need to be integrated. The schema alignment process is usually done in two steps [1]:

- *Schema matching step* – identification of semantically related attributes from data sources' schemas. Examples include "attribute *name* in one source corresponds to attribute *title* in another", and "*location* is concatenation of *city*, *state*, and *zip code*".
- *Schema mapping step* – derivation of rules that specify how to translate data across schemas. Mappings are typically structured queries written in a language such as SQL.

In practice, creating the matches and mappings is not a trivial task because it requires understanding of the semantics of the schemas of the data sources and thus consumes quite a bit of the effort in setting up a data integration application. Schema matching and mapping is a well-studied topic in the database, machine learning, and Semantic Web communities (see [149][56] for surveys). Various solutions on semi-automatic schema mapping range from schema-only string similarity techniques to rule-based methods to application of machine learning algorithms, which also involve data instances (e.g. Label Propagation [172]). Modern schema matchers combine output from multiple sub-matchers [172][130][150]. However, the task is inherently a heuristic one and there is no algorithm that will take two arbitrary schemas and flawlessly produce correct matches and mappings between them.

2.2 KEYWORD SEARCH: INTEGRATION ON DEMAND

Another group of related works involves keyword search in single-database (e.g., BANKS [21], DBXplorer [5], Discover [88]) and peer-to-peer contexts (e.g., [131][197]).

Keyword search allows users to post a query as a set of keywords which match schema or data values without specifying which sources to look at. Even though different approaches to keyword search use different techniques, at the core they all deal with the problem of searching a graph to find all sub-graphs that satisfy certain properties. The problem is NP-hard and known to be exponential in the query size, thus making query execution prohibitively expensive [13]. To improve query performance, different heuristics have been proposed. For example [95] proposes to reuse and combine computation across queries.

A number of works are available on top-*k* query processing (e.g., [30][116][178][170]) that returns only top-*k* result according to some scoring function. Another way is to produce answers that can be generated quickly and then to provide users with query forms that characterize the unexplored portion of the answer space [13].

2.3 CROWDSOURCING IN DATABASES AND DATA INTEGRATION SYSTEMS

An important component of the architecture that I propose exploits the advantages offered by crowdsourcing applications and collective intelligence. Successes of crowdsourcing systems, for example Wikipedia and Linux, inspired the idea to apply crowdsourcing for large-scale data integration. So far in data integration systems, crowdsourcing was used mostly as external resource to perform separate work for helping answering queries [169][70][140][141], checking schema mapping [128][16][142], or for entity resolution [183][50]. However, to the best of my knowledge, none of existing works make crowdsourcing techniques as a central and internal component for the advanced data integration.

In any crowdsourcing environment quality verification and assurance operations are required. The amount of research in the area of data conflict resolution and querying inconsistent data is considerable. See [23][20] for a comprehensive review of the current state of the art. Conflicts can be resolved using metadata about data source accuracy and freshness, or exploiting dependencies between data sources, where information from one source can be re-used in another source. Data inconsistency as a key integrity constraint violation was considered in [4]. Consistent query answering that ignores inconsistent data, thereby violating integrity constraints, was introduced in [31]. This approach is related to more recent research on query transformation for consistent query answering [186]. An alternative approach is based on inconsistent database repair, producing a minimally different – yet consistent – database that satisfies integrity constraints [25].

2.4 RELATED DATA INTEGRATION/CURATION SYSTEMS

While much research addresses different data integration subtasks, to the best of my knowledge, no existing system implements all stages of data integration processes in an advanced infrastructure based on collective intelligence. The closest related work is Orchestra with Q systems [94][175][170][93][172][95]. Orchestra is a Collaborative Data Sharing Systems focused mostly on data exchange and update reconciliation similar to version control systems. With Q system users can post keyword queries to the system and as the result obtain “best” *k* records. To find the best records, authors propose a machine-learning algorithm to incorporate users’ feedback (in form of like and dislike votes) into computation of schema mapping confidence [170]. A significant disadvantage of the Orchestra system is its long set-up time. Another system, Data Tamer [166][77][173], showed that combination of machine learning techniques and crowdsourcing allows to significantly reduce the amount of work and time needed to perform data curation tasks. However, Data Tamer workflow assumes a specific customer with number of datasets with a dedicated Data Tamer Administrator and crowdsourcing is limited to the set of domain experts. Both Orchestra and DataTamer don’t have a goal to create a global-scale interdisciplinary integrated repository and instead of supporting true crowdsourcing architecture in a Wikipedia-like manner they utilize either domain experts or limit users to actions to only a few functions.

2.5 DATA REPOSITORIES

A number of tools (e.g., DataUp [168]) and data repositories (e.g., ONEShare [168], Dataverse Network [106], DataDryad [58], DSpace [165], Dash [46]) were developed to facilitate data sharing and preservation processes. Usually a data repository is a cloud service with a web interface that allows users to submit data via a web browser.

Advantages of data repositories include ease of use, persistent storage, public distribution and recognition (through citation via unique dataset identifier), and search for datasets based on metadata. Some repositories provide visualization and statistical analysis tools.

The disadvantages of current approaches include *repository isolation* and *dataset isolation within a repository*. The former disadvantage is related to the fact that some repositories are created only for specific research areas, journals or universities. Therefore, users would need to know where to find the dataset they are interested in and where to submit their dataset. Dataverse Network and Databib [48] attempt to solve the problem by allowing users to search within a set of repositories *but on dataset level metadata only*: the first one does it automatically as all Dataverse networks are connected, and the second one allows users to create and curate records that describe data repositories that users can search. Also Open Archives Initiative [110][139] has developed OAI-PMH [138] specification for repository metadata harvesting which provides low-barrier mechanism for repository interoperability.

However, the latter disadvantage – dataset isolation – has not been resolved in any of the existing data repositories. Most existing data repositories do not actually process the data files submitted to them and thus cannot establish any relationships between datasets on a variable level.

To the best of my knowledge, the problem of efficient and reliable global-scale information integration has not been systematically addressed.

3.0 ADVANCED DISTRIBUTED DATA INTEGRATION INFRASTRUCTURE UNDER MAGNIFYING GLASS

In this chapter I describe in detail some of the most important components of the infrastructure that addresses the research questions mentioned in Section 1.3. Development of each component is based on the reviewed literate in Section 2.0. Where possible I explicitly state how my solutions differ from existing approaches and provide reasons why I do it that way.

First of all, the infrastructure should allow users to submit their datasets with some dataset level metadata. The dataset level metadata includes title and description of the dataset so that other users can browse the repository. We store dataset level metadata in a *source_info* table. The schema for the *source_info* table can be designed without knowing in advance what kind of data users are going to submit, since it stores general information about the dataset. More interesting question is how to ingest and store heterogeneous datasets, whose schema vary and is unknown in advance. I developed two approaches for storing heterogeneous datasets, which I discuss in Section 3.1. Once the datasets are stored successfully, they need to be integrated. Section 3.2 discusses the virtual integration model that I developed based on notion of discovering and maintaining relationships between datasets (similar to P2P approach from Section 2.1.2.1) instead of actually transforming and merging all datasets into one integrated dataset (e.g. Data Warehouse from Section 2.1.1.1). Finally I discuss novel techniques to search throughout integrated datasets in Section 3.3.

3.1 STORING HETEROGENEOUS DYNAMICALLY INCOMING DATASETS

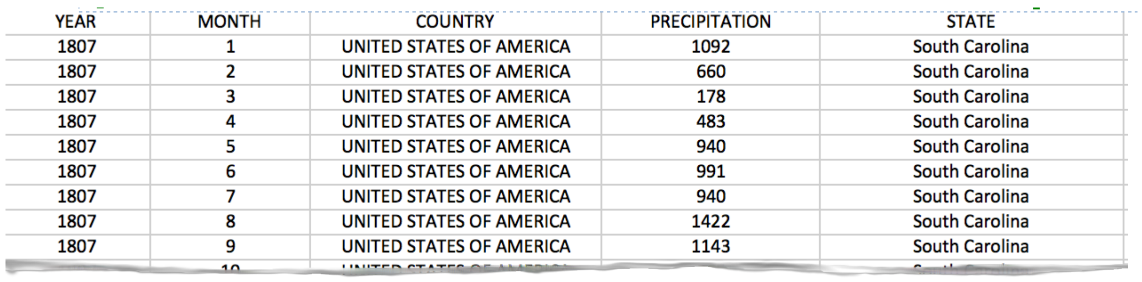
Initially, my proposed infrastructure does not have any data and one of the functionality that it provides to its users is the data ingest. Data ingest allows users to submit their datasets to the system. The datasets that users have are heterogeneous with respect to the following three features:

- File format – different file types.
- Schema – attributes that refer to the same entity may have different names and types.
- Values – values that refer to the same entity have different representation.

In order to handle file format heterogeneity, we need to develop extractors that extract data from different files to a common format. A separate challenge is how to store datasets with various schemas.

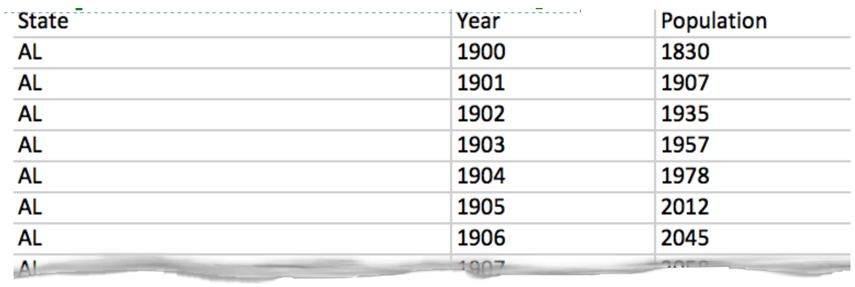
Datasets are generated independently and the intent of data usage in various contexts is unknown at the system development time. Designing strict target schema and requiring users to transform their datasets to conform to that schema is not a feasible solution. Strict target schema will inevitably result in data loss during transformation and inability to store some datasets.

Below I discuss two alternative approaches. For the explanation of the proposed approaches I consider the two datasets D_3 and D_4 that were introduced in the Section 1.1 and are shown on Figure 5 and Figure 6 respectively.



YEAR	MONTH	COUNTRY	PRECIPITATION	STATE
1807	1	UNITED STATES OF AMERICA	1092	South Carolina
1807	2	UNITED STATES OF AMERICA	660	South Carolina
1807	3	UNITED STATES OF AMERICA	178	South Carolina
1807	4	UNITED STATES OF AMERICA	483	South Carolina
1807	5	UNITED STATES OF AMERICA	940	South Carolina
1807	6	UNITED STATES OF AMERICA	991	South Carolina
1807	7	UNITED STATES OF AMERICA	940	South Carolina
1807	8	UNITED STATES OF AMERICA	1422	South Carolina
1807	9	UNITED STATES OF AMERICA	1143	South Carolina

Figure 5: Dataset D3



State	Year	Population
AL	1900	1830
AL	1901	1907
AL	1902	1935
AL	1903	1957
AL	1904	1978
AL	1905	2012
AL	1906	2045

Figure 6: Dataset D4

3.1.1 "One table" approach

One way to store the datasets without knowing their schemas in advance is by turning them into key-value pairs and then storing those pairs in one table. A key is an attribute name and value is the value of that attribute. Each key-value pair is annotated with a dataset identifier. To reconstruct the original table, each pair is also annotated with tuple-id. Thus, each tuple in the resulting table has four elements (*id*, *i*, *key*, *value*), where *id* is the dataset id, *i* is the tuple-id, *key* is the attribute name and *value* is the attribute value of *i*-th tuple. The algorithm that converts input dataset to the one table format is called CONVERTTOONETABLESTORAGE and is shown in Algorithm 1.

Algorithm 1: CONVERTTOONETABLESTORAGE(*S, D, id*). **I**nput: Schema *S*, list of tuples *D*, id of the datasets *id*. **O**utput: List of tuples in one table format.

```
1:  $R \leftarrow$ empty list
2:  for $i = 0$ to LENGTH($D$) do
3:      for $j = 0$ to LENGTH($S$) do
4:          $key = S[j]$
5:          $tuple = D.GET(i)$
6:          $value = tuple[j]$
7:          $R$.APPEND(($id$, $i$, $key$, $value$))
8:      end for
9:  end for
10: return $R$
```

The examples of applying CONVERTTOONETABLESTORAGE to datasets D_3 and D_4 are shown on Figure 7 and Figure 8 respectively. Both datasets after transformation have the same schema and can be easily stored in one table.

The "one table" approach is simple and intuitive; however, it suffers from several disadvantages. First disadvantage is that large number of duplicate values that need to be stored results in storage overhead. Each attribute of the schema is stored repeatedly for each value of that attribute. In addition, dataset id and tuple-id need to be stored with each key-value pair. As the result, the required number of cells to store a table is *four* times larger that the original table (since additional three pieces of information are stored for each value). We could decompose the table and store attribute names in a separate table, but then it would not be a "one table" approach and would require an expensive join operation.

D_3

id	i	key	value
3	1	YEAR	1807
3	1	MONTH	1
3	1	COUNTRY	UNITED STATES OF AMERICA
3	1	PRECIPITATION	1092
3	1	STATE	South Carolina
3	2	YEAR	1807
3	2	MONTH	2
3	2	COUNTRY	UNITED STATES OF AMERICA
3	2	PRECIPITATION	660
3	2	STATE	South Carolina

…

Figure 7: Example of dataset D_3 converted into one table format

D_4

id	i	key	value
4	1	State	AL
4	1	Year	1900
4	1	Population	1830
4	2	State	AL
4	2	Year	1901
4	2	Population	1907

…

Figure 8: Example of dataset D_4 converted into one table format

The second disadvantage is that if we want to show the whole table back to the user, we would need to convert the data from key-value pairs back to the original view. We found that this operation introduces additional overhead. Algorithm 2 shows CONVERTFROMONETABLESTORAGE algorithm that converts tuples from one table format to the original relational format.

Algorithm 2: CONVERTFROMONETABLESTORAGE(*S*, *Rs*). **I**nput: Schema *S*, list of tuple in “one table” format for one dataset sorted by tuple-id *Rs*. **O**utput: List of tuples in one table format.

```
1:  A ← empty list
2:  numAtt = LENGTH(S)
3:  j = -1
4:  resTuple = () //empty tuple
5:  for i = 0 to LENGTH(Rs) do
6:      j = j + 1
7:      tuple = Rs.GET(i)
8:      resTuple[j] = tuple[3] //3 is the index of the value in the tuple
9:      if j = numAtt – 1 then
10:         A.APPEND(resTuple)
11:         resTuple = ()
12:         j = -1
13:     end if
14: end for
15: return A
```

Third disadvantage of the “one table” approach is indexing. The index on that table will grow fast, which will impact the lookup performance.

3.1.2 “A database per dataset” approach

Another way to store heterogeneous datasets is to store them separately and maintain system-wide catalog with all the required metadata that will allow system to operate. “A database per dataset” approach does exactly that - each dataset is stored in a separate database.

A dataset might actually be a collection of two or more data tables that are strongly related to each other. For example, an excel file with several sheets or a relational database with several tables, where some sheets/tables refer to other sheets/tables. “A database per dataset” approach naturally supports this model and provides a way to keep the related data tables together and to form a namespace.

This approach is intrinsically distributed since it allows us putting the data on different machines. This architecture is similar to Hadoop Distributed File System [162] where one node, called *namenode*, maintains all the metadata information and knows where a particular dataset is stored. The nodes that store the actual data are called *datanodes*. In my proposed architecture the *namenode* is called *metadata* node and the *datanodes* are also called *datanodes*.

The minimum metadata information that needs to be maintained for each datasets is the connection information to the database that stores it. Additionally, storing variable-level metadata (such as name, description, type, etc.) from all datasets in the metadata node will allows us to establish relationships between datasets, as explained in Section 3.2.1.

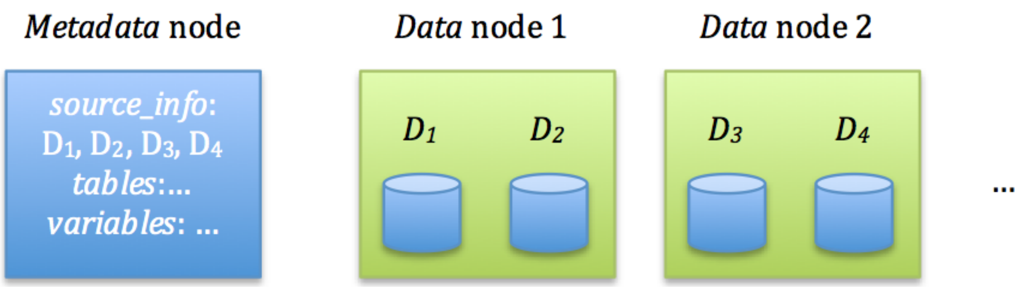


Figure 9: “A database per dataset” architecture with one *metadata* node and many *datanodes.* *Metadata* node stores connection information to the databases that store the data on *datanodes*

Figure 9 shows a simplified configuration of the “A database per dataset” approach for storing the datasets from the motivation example in Section 1.1. The dataset D_1 is stored in database D_1 and the dataset D_2 is stored in database D_2 on the *datanode* 1, whereas the dataset D_3 is stored in database D_3 and dataset D_4 is stored in database D_4 on the *datanode* 2. The *metadata* node stores only metadata for all datasets such as titles, descriptions, table names, variable names and connection information for the databases on the *datanodes*.

Compared to “One table” approach, this approach does not require any transformation of the dataset and has no storage overhead due to the repetition. The only overhead is the time to create new databases and tables for new datasets, however this only needs to be done once and it is relatively inexpensive operation. In addition, individual databases can be easily replicated on several nodes to ensure high availability in case of node failures.

The *metadata* node is the single point of failure in this architecture. If the metadata node goes down, the whole system cannot operate even though the data are available on the data nodes. There are several ways to ensure high availability even in case of *metadata* node failures. For example, a secondary *metadata* node can run on a different machine that will keep up with the current state of the main *metadata* node and be ready to substitute it in case of failure.

Based on the advantages and disadvantages of the two approaches discussed above, I selected the “A database per dataset” approach as the storage solution for the heterogeneous datasets in the proposed infrastructure. The rest of the document assumes that we use the “A database per dataset” approach.

The proposed solution for storing previously unknown heterogeneous datasets can be used for archival purposes as a data repository similar to the existing ones reviewed in Section 2.5. Since we are interested in creating an *integrated* repository, in the next section I elaborate on data integration methods.

3.2 INTEGRATING DATASETS

As explained in the previous sections, the datasets are stored in separate databases and possibly distributed among several machines. Such setup is similar to P2P system where each database is a peer. In contrast to P2P systems, my infrastructure implements a metadata node that maintains information about all datasets in the system. The next step is to integrate those datasets.

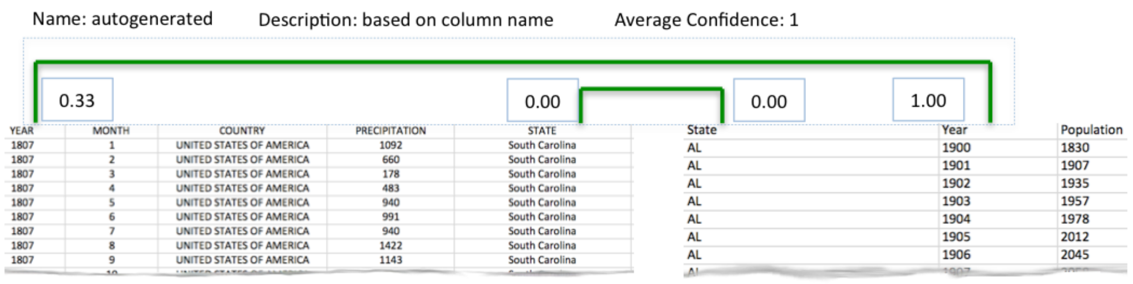
Note that since we do not know the context or all scenarios of how the datasets might be used we would like to preserve datasets autonomy that would allow them to evolve independently while maintaining connection to other datasets. In Section 3.2.1 I elaborate on the process of discovering how datasets are related to each other. In Section 3.2.2 I describe how the information about relationships between datasets is represented. In Section 3.2.2.2 I present conceptual and physical models that reflect how relationships are maintained.

3.2.1 Discovering relationships

In order to integrate datasets, we need to apply schema matching and schema mapping algorithms for all pairs of the datasets in our repository. I call processes of schema matching and schema mapping as *relationship discovery*. The relationship discovery process between two datasets results in the information that tells us how those two datasets are related to each other. I call that information a *relationship*. I elaborate more on the relationship model in the Section 3.2.2.

Currently a relationship between datasets is established only based on linguistic similarity between variables' metadata (such as variable name or description). For example, Figure 10 shows an example of a relationship between two datasets D_3 and D_4 based on same names of the two variables in both datasets. The schema matching occures between $D_3.YEAR$ and $D_4.Year$, and

$D_3.STATE$ and $D_4.State$. The schema mapping results in an equality operation, i.e. $D_3.YEAR = D_4.Year$, and $D_3.STATE = D_4.State$. The values 0.33, 0.00 and 1.00 are the data overlapping values that will be described in the next section together with more details on what metadata constitutes a relationship model.


Name: autogenerated Description: based on column name Average Confidence: 1

YEAR	MONTH	COUNTRY	PRECIPITATION	STATE
1807	1	UNITED STATES OF AMERICA	1092	South Carolina
1807	2	UNITED STATES OF AMERICA	660	South Carolina
1807	3	UNITED STATES OF AMERICA	178	South Carolina
1807	4	UNITED STATES OF AMERICA	483	South Carolina
1807	5	UNITED STATES OF AMERICA	940	South Carolina
1807	6	UNITED STATES OF AMERICA	991	South Carolina
1807	7	UNITED STATES OF AMERICA	940	South Carolina
1807	8	UNITED STATES OF AMERICA	1422	South Carolina
1807	9	UNITED STATES OF AMERICA	1143	South Carolina

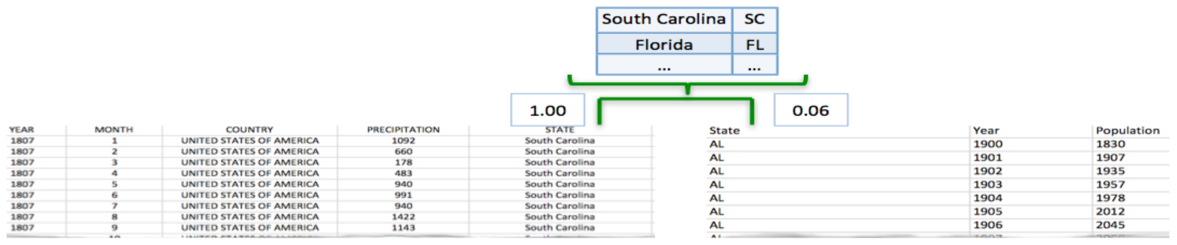
State	Year	Population
AL	1900	1830
AL	1901	1907
AL	1902	1935
AL	1903	1957
AL	1904	1978
AL	1905	2012
AL	1906	2045

Figure 10: Automatically discovered relationships between two datasets D_3 and D_4 based on same names of the two variables in both datasets

As it was mentioned in the Section 2.1.3, many different schema matchers that incorporate multiple features and use advanced machine learning algorithms have been developed in recent years. Thus, I have developed an extendable architecture that enables easy way to utilize different schema matching and schema mapping tools.

As noted in [83], automatic schema mapping is often not foolproof. Therefore, I propose a novel approach combining automatic relationship discovery with crowdsourcing techniques. In addition to submitting datasets, users can provide feedback on automatically discovered relationships and/or create relationship manually. The feedback is provided in terms of confidence values ranging from 0 to 1 that reflect users' belief that relationships hold. When creating new relationships, users have to identify schema matching and specify schema mapping manually.

Schema mappings can be more complex than simple equality operation and may involve data transformation performed automatically or defined by users. An automatic transformation is based on variable metadata or can be selected by users from predefined list of transformations. For example, consider date conversion from DD-MM-YYYY to MM-DD-YYYY format, or measurement unit transformation from miles to kilometers. A user-defined transformation is either a mathematical or string manipulation expression on datasets variables, e.g. CONCAT(street, city, state) to concatenate street, city and sate values into one value, or definition of correspondence tables (I call them synonyms transformation). Figure 11 shows an example of the synonyms transformation applied to the $D_3.STATE$ and $D_4.State$ variables. The transformation defines correspondence table between US state's full name and its two letter abbreviations.



South Carolina	SC
Florida	FL
...	...

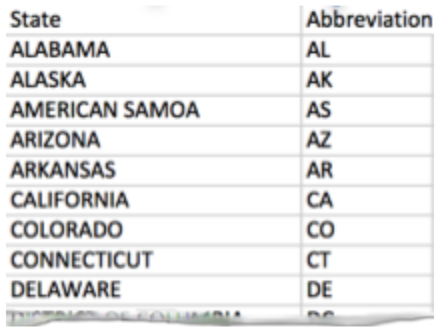
YEAR	MONTH	COUNTRY	PRECIPITATION	STATE
1807	1	UNITED STATES OF AMERICA	1092	South Carolina
1807	2	UNITED STATES OF AMERICA	660	South Carolina
1807	3	UNITED STATES OF AMERICA	178	South Carolina
1807	4	UNITED STATES OF AMERICA	483	South Carolina
1807	5	UNITED STATES OF AMERICA	940	South Carolina
1807	6	UNITED STATES OF AMERICA	991	South Carolina
1807	7	UNITED STATES OF AMERICA	940	South Carolina
1807	8	UNITED STATES OF AMERICA	1422	South Carolina
1807	9	UNITED STATES OF AMERICA	1143	South Carolina

State	Year	Population
AL	1900	1830
AL	1901	1907
AL	1902	1935
AL	1903	1957
AL	1904	1978
AL	1905	2012
AL	1906	2045

Figure 11: Example of synonyms transformation between $D_3.STATE$ and $D_4.State$ variables. The transformation defines correspondence table between US state's full name and its two letter abbreviations

One disadvantage of synonyms transformation appears when there is a large number of distinct values that are not matching. Especially if those mappings are well known and might be available (e.g., US full state names to two letter abbreviations mapping is well known information). This situation is not an unusual case in the proposed infrastructure. Consider an example in Figure

12. Figure 12 shows a dataset D_2, from the motivation example in Section 1.1, that contains two columns: State and Abbreviation, defining the mappings between full names of the US states and their two letter abbreviations. When D_2 is submitted to the repository, the relationships between all three of the example datasets (D_2, D_3 and D_4) will be discovered automatically as shown on the Figure 13. The relationship between variables $D_2.Abbriviation$ and $D_4.State$ can be either discovered automatically based on the similarity in variables' descriptions and/or values, or can be added manually by users. Now we do not need to define the correspondence table and the datasets D_3 and D_4 are related to each other via dataset D_2.



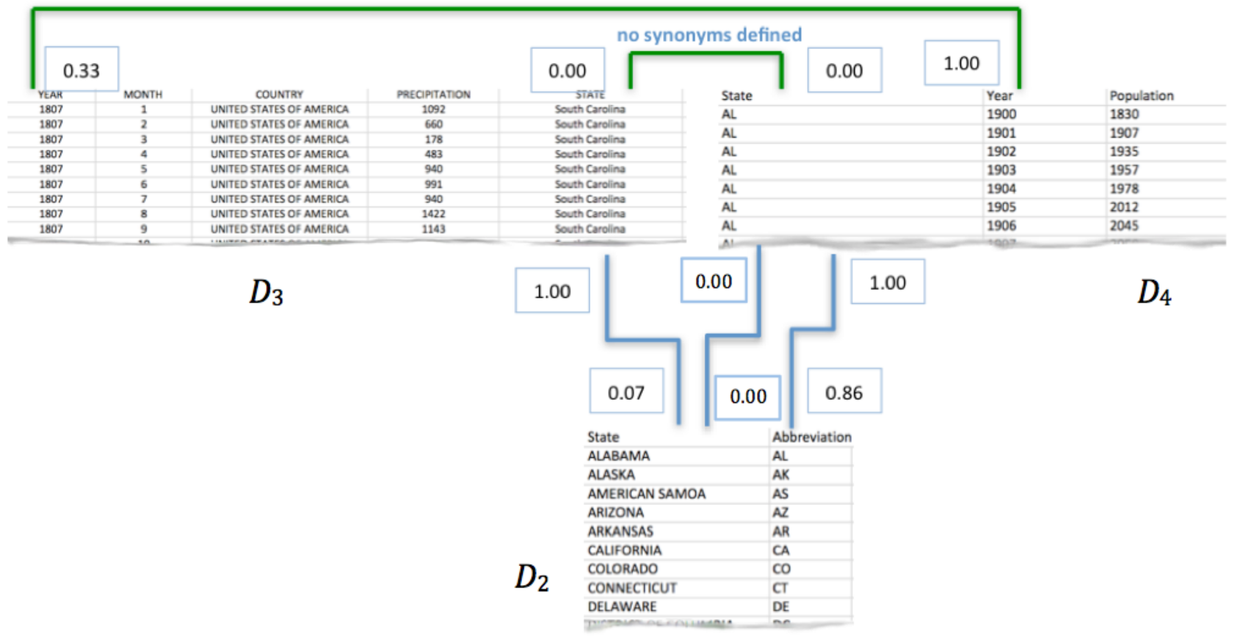
State	Abbreviation
ALABAMA	AL
ALASKA	AK
AMERICAN SAMOA	AS
ARIZONA	AZ
ARKANSAS	AR
CALIFORNIA	CA
COLORADO	CO
CONNECTICUT	CT
DELAWARE	DE
[illegible]	[illegible]

Figure 12: Example of dataset D_2

The origins of all three datasets can be completely independent, e.g. they may result from research in remotely related disciplines and be submitted by users who are not aware of each other. In general, two datasets can be related to each other via several other datasets. Figure 14 shows generalized schematic example of relationships between 6 datasets (heterogeneity is shown by different shapes and colors that represent difference in datasets file formats, or schema, or values). If you simply look at the datasets D_5 and D_{10} outside of the relationships context, you might first think that they are not related to each other. However, they are related via other datasets, e.g. D_5 −

$D_7 - D_9 - D_{10}$, or $D_5 - D_8 - D_9 - D_{10}$. Such transitive relationships will be utilized later when we will consider the data exploration and keyword search in Section 3.3.

Next section describes relationship model in more details.



YEAR	MONTH	COUNTRY	PRECIPITATION	STATE
1807	1	UNITED STATES OF AMERICA	1092	South Carolina
1807	2	UNITED STATES OF AMERICA	660	South Carolina
1807	3	UNITED STATES OF AMERICA	178	South Carolina
1807	4	UNITED STATES OF AMERICA	483	South Carolina
1807	5	UNITED STATES OF AMERICA	940	South Carolina
1807	6	UNITED STATES OF AMERICA	991	South Carolina
1807	7	UNITED STATES OF AMERICA	940	South Carolina
1807	8	UNITED STATES OF AMERICA	1422	South Carolina
1807	9	UNITED STATES OF AMERICA	1143	South Carolina

State	Year	Population
AL	1900	1830
AL	1901	1907
AL	1902	1935
AL	1903	1957
AL	1904	1978
AL	1905	2012
AL	1906	2045

State	Abbreviation
ALABAMA	AL
ALASKA	AK
AMERICAN SAMOA	AS
ARIZONA	AZ
ARKANSAS	AR
CALIFORNIA	CA
COLORADO	CO
CONNECTICUT	CT
DELAWARE	DE

Figure 13: Example of the relationships between three datasets D_2, D_3 and D_4

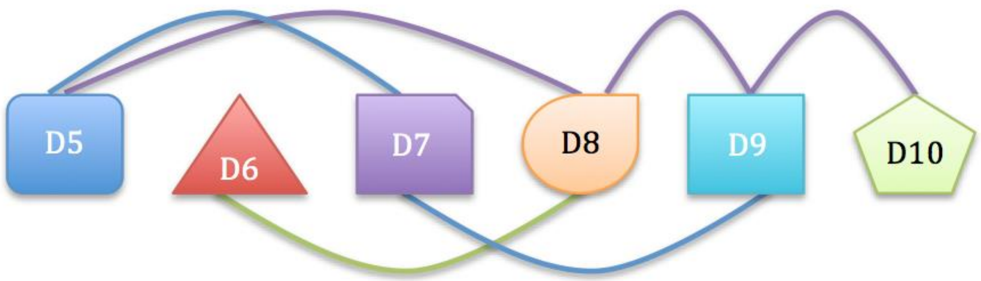


Figure 14: Generalized schematic example of the data integration based on relationships discovery

3.2.2 Relationship model

Each relationship consists of one or more links that represent an actual matching and mapping between variables from two datasets. Conceptually, a link reminds a foreign key in relational data model but with no strict referential integrity constraint. Since the link does not define which datasets is referencing and which dataset is referenced, both datasets have equal roles. Thus, a dataset on the left end of the link is called the left dataset and a dataset on the right end of the link is the right dataset. In Figure 10, the example relationship consists of two links: one link is $D_3.YEAR = D_4.Year$, and the other link is $D_3.STATE = D_4.State$. D_3 is the left dataset and D_4 is the right dataset.

Each link is associated with two data overlapping values at its ends that are explained below in subsection 3.2.2.1.

3.2.2.1 Data Overlapping values Data Overlapping (DO) value on each end of a relationship link shows how many values from the dataset on that end were matched with values from the dataset on the other end of the link.

In Figure 10, $DO_{D3.YEAR} = 0.33$ and $DO_{D4.Year} = 1$ mean that only 33% of values in the $YEAR$ column in the dataset D_3 are matching values of the $Year$ column in the dataset D_4, whereas 100% of the values in the $Year$ column in the dataset D_4 are matching the values in the $YEAR$ column in the dataset D_4. The explanation for that is that the dataset D_4 has records only for 20th century, whereas D_3 has records for 19th, 20th and 21st centuries. $DO_{D3.STATE} = 0$ and $DO_{D4.State} = 0$ simply mean that none of the values in $STATE$ column in dataset D_3 are matching values in $State$ column in dataset D_4 and vice versa because $D_3.STATE$ contains full names of the states in US whereas $D_4.State$ contains two letter abbreviations. When we defined the synonyms transformation between $D_3.STATE$ and $D_4.State$ variables in Figure 11, the DO values are also recalculated automatically showing that all values in the $D_3.STATE$ match the $D_4.State$ variable, whereas only 6% values were matched vice versa. The explanation is that the $D_3.STATE$ dataset has records only for 3 states, whereas $D_4.State$ has records for all 50 states of the US.

The DO value less than 1 means that not all values were matched and this might happen due to two reasons. First, the other dataset does not contain some values or, second, those values are represented in different way. In the former case there it nothing can be done to fix it, whereas in the latter case record level linkage (synonyms transformation) need to be performed. Record level linkage can be done either manually by creating a concordance table, or by using entity reconciliation techniques [45][17][42][63][75].

DO values are calculated according to Algorithm 3. As an input, the algorithm accepts left and right sides of the link's schema mapping expression. First the left expression is decoded (line 1) into dataset's identifier (D), list of columns (Cl) and the transformation expression (Tr). Then (line 2) the values of the columns Cl are transformed according to the transformation expression Tr. Line 3 and 4 do the same operations for the right side of the schema mapping. To find matching values, line 5 performs join between transformed left and right values. Any join technique can be used here. Line 6 and 7 simply calculate DO values as the ratio of matched value in each side and line 8 returns both DO values.

Algorithm 3: DATAOVERLAP(*left*, *right*). **I**nput: Left side of the schema mapping expression *left*, right side of the schema mapping expression *right*. **O**utput: DO values of left and right parts of the schema mapping.

```
1:  $(D, Cl, Tr)$ = DECODE($left$)
2:  $L$ = TRANSFORM($D, Cl, Tr$)
3:  $(D, Cl, Tr)$ = DECODE($right$)
4:  $R$ = TRANSFORM($D, Cl, Tr$)
5:  $J$ = JOIN($L$, $R$)
6:  $lDO$ = LENGTH($J$) / LENGTH($L$)
7:  $rDO$ = LENGTH($J$) / LENGTH($R$)
8:  return $(lDO, rDO)$
```

3.2.2.2 Relationship's Data Overlapping value To describe the quality of a relationship in terms of DO values of the links that constitute the relationship, a relationship DO value is calculated using the equation (1).

$$DO = max\left(\frac{1}{n}\sum_{i=1}^{n}\left(link_{i,lDO}\right), \frac{1}{n}\sum_{i=1}^{n}\left(link_{i,rDO}\right)\right) \tag{1}$$

where n is a number of links in the relationship for which DO value is being calculated; $link_{i,lDO}$ is the DO value of the i-th link's left side; $link_{i,rDO}$ is the DO value of the i-th link's right side.

To conclude on the Relationship Model, each relationship is associated with some static and dynamic metadata. Static metadata includes name, description, creator of the relationship, and time and date when it was created. Dynamic metadata includes average confidence value, the total number of feedbacks, and relationship DO value.

3.2.3 Schema Graph

3.2.3.1 Conceptual Model Schemas of tables of data sources and discovered relationships between them are represented as a global *Schema Graph* – an undirected multigraph where nodes represent data tables and edges represent relationships between tables. When a new dataset is added, Schema Graph is expanded to include the schemas of the data tables from the dataset as new nodes in the graph and discovered relationships are added as edges. Figure 15 shows conceptual Schema Graph for the four datasets from Section 1.1 (variable are represented with first letter).

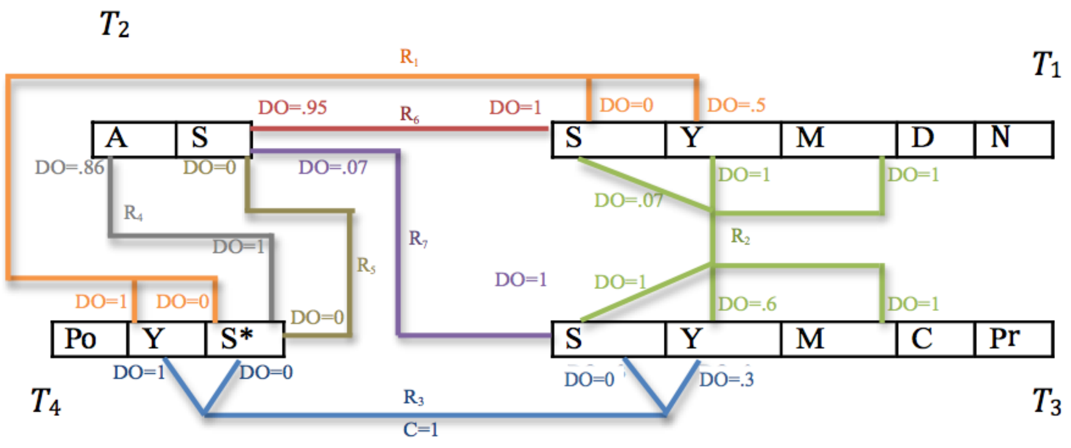


Figure 15: Excerpt of conceptual Schema graph for four datasets from Section 1.1 (variable are represented with first letter)

3.2.3.2 Physical Model The physical model of the schema graph describes the way schema graph is implemented in our infrastructure. Part of the schema graph is stored in a relational store as apart of the metadata database and part of it is stored in a graph store located on the *metadata* node (but technically can be run on a separate machine).

Figure 16 illustrates how the schema graph is separated between two stores. Relational store contains all relationship metadata, foreign keys to the dataset metadata as well as information about links that constitute the relationship. The graph store maintains the schema graph $G = (V, E)$. Set of nodes $V(G) = \{t_1, \cdots, t_n\}$ represent data tables where $t_i \in V(G)$ is the id of the table. Set of edges $E(G) = \{r_1, \cdots, r_m\}$ represent relationships between tables where $r_i \in E(G)$ is the id of the relationship.

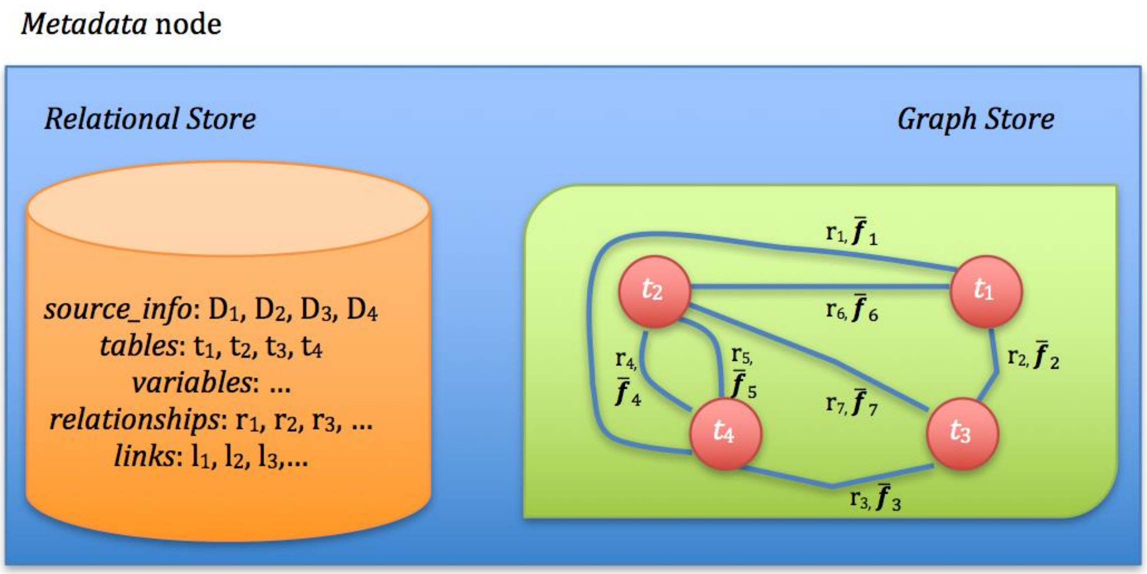


Figure 16: Schematic illustration of Schema Graph physical model

3.2.3.3 Edge Feature Vector based Cost Model Each edge $r_i \in E(G)$ of the schema graph is associated with a feature vector $\bar{\boldsymbol{f}}_i$ that specifies the values of all the features of the edge. Features encode the aspects of edges that are relevant to ranking of queries that will be discussed in Section

3.3. Essentially, they capture distinctions that may be relevant to a user's preference for an edge as part of the query. Currently, (1 - average confidence) and ($1 - DO$) values are used as features. However, other features can be easily added. For example, let average confidence value be 1 and DO equals 0.5 for relationship with id r_3, then $\bar{\boldsymbol{f}}_3 = [0, 0.5]$. Each feature has a weight, representing the relative contribution of that feature to the cost of the edge. Initially, the weights are normally distributed between features. Later users may configure them depending on the task at hand (more on this is in Section 3.3.4).

The costs associated with edges in the schema graph are simple weighted linear combinations of edge feature vector calculated using equation (2).

$$C_i = \bar{\boldsymbol{w}} \cdot \bar{\boldsymbol{f}}_i \tag{2}$$

where C_i is the cost of the edge for the relation r_i, $\bar{\boldsymbol{f}}_i$ is the feature vector of the edge for the relation r_i, and $\bar{\boldsymbol{w}}$ is the vector of weights for each feature.

3.3 EXPLORING THE REPOSITORY

So far we have covered how heterogeneous datasets are stored (Section 3.1) and how they are integrated (Section 3.2). Next we discuss how the repository can be used, e.g. how can we find relevant datasets and, more importantly, how can we find the data that we are interested in. Thus, the next main functionality block is the repository exploration.

Even though the datasets can be stored on many different machines, all metadata is stored in the single place on the metadata node. Thus, searching for datasets based on their metadata (such as title, description, user who submitted, etc.) is a straightforward functionality to implement. In fact, all data repositories provide such functionality. The result of such search is usually a list of datasets that might have data that user is interested in. The user is then required to open each dataset separately and try to integrate the data to perform any analysis. This is similar to a web search engine that accepts as input several keywords and returns the list of links to the web sites that contain those keywords, but the user is required to open each link and to find the information he or she needs on each web page.

More useful functionality would be to allow users to search for particular variables throughout the repository without specifying (and knowing in advance) which datasets those variables come from. The result of such search would also be not just a list of datasets, but a table with data integrated from the relevant datasets. The user would not need to worry how to integrate the data and would be able to focus on the data analysis to answer his or her research questions.

As it was mentioned in the Section 2.2, several works considered keyword search in single database, where the schema is known in advance. Some on them were focused on keyword search in P2P environments, others on top-k keyword search. The closest system that implements similar approach to keyword search is the Q system [170]. However, all previous approaches either worked with existing schema and/or focused only on finding top-k records. In contrast, we do not try to find only top-k highest-scoring answers. Instead, we find datasets and integrate them on the fly.

Currently I focus only on keyword search where keywords are variable names. However, the approach can be extended to include variables and dataset level metadata.

In this section, I explain how the keyword search over the variables throughout repository works in the proposed infrastructure. I begin with formalizing the keyword search in Section 3.3.1. In Section 3.3.2, I describe how keywords match against schema graph and how that results in trees over graph. In Section 3.3.3, I show how trees over schema graph are converted into SQL queries. Finally, in Section 3.3.4, I show how the queries are executed and what is returned as the result of the keyword search.

3.3.1 From Research Question to Keyword Search

Let us go back to the research question from the motivation example (Section 1.1):

"*Is there a correlation between population and number of disease cases in United States*?"

To answer this question, we would need the data from two distinct datasets:

- A dataset that contains population numbers in US (a census data could be a good source).
- A dataset that contains diseases information (number of cases) in US (historical medical records could be a good source).

The research question can be formulated as the following keyword query:

population, "*number of disease cases*"

A user does not need to know and specify which datasets to look at, he or she simply poses the keyword query and the system searches throughout the whole repository.

3.3.2 From Keywords to Trees

Given a set of keywords $Q = \{q_1, \cdots, q_k\}$, the goal is to find one or more sequences of tables that can be merged together to answer the query. For that, two tasks need to be performed:

- identify the tables that contain the keywords;
- traverse existing relationships between those tables to find sequences of tables that need to be integrated.

The first task is relatively easy since all metadata (including all variable names from all datasets) is stored in one place – relational store on *metadata* node. Thus, we can find a set of tables that contain the variables of interest by executing SQL equivalent of the relational algebra expression (3).

$$T = \pi_{did}\left(tables \bowtie \sigma_{name=q_1 \vee \ldots \vee name=q_k}(variables)\right) \tag{3}$$

where $\sigma_{name=q_1 \vee \ldots \vee name=q_k}(variables)$ selects only those variable where variable name exactly matches one of the keyword from the Q; $T = \{t_1, \cdots, t_l\}$, where t_i – is the id of the table that contains one or more $q \in Q$. The exact match condition can be relaxed by either looking for substring or similarity match between variable name and keywords. For the *population*, "*number of disease cases*" keyword search, $T = \{t_1, t_4\}$.

Conceptually, T represents a subset of nodes $N \subseteq V(G)$ in the schema graph G that matched keywords (see Figure 17 for schema graph with nodes that matched keyword nodes in black and calculated edge costs). Thus, the second task is to utilize schema graph, graph store and graph algorithms to find subgraphs of G that connect nodes from N. Formally, the problem of determining the closest interconnections between two, three, or more nodes in a graph is the Steiner tree problem [89].

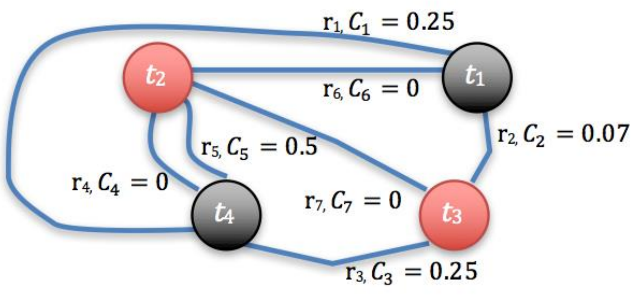


Figure 17: Schema graph with costs; black nodes matches keywords

Prior related work focused on finding either a single tree or k lowest-cost trees that contain ***all*** of the keyword nodes and only worked on connected graphs. Since in our case datasets and relationships are added dynamically and datasets are from multiple domains, we cannot make an assumption that schema graph is always (or ever) a connected graph. In addition, most of the previous work focused on finding top-k answers to the query, whereas in Q [170] and our infrastructure the goal is to find one or more trees that will then lead to the datasets that need to be integrated. Steiner-tree problem is NP-hard [13], hence it is not feasible to use exact Steiner tree algorithms over a large graph. Many approximation and heuristics-based algorithms to solve Steiner tree problem were developed (e.g. [170][100][129][2][99][57][54][85][90][109]). The most popular is backward expansion/distance network heuristic that in the first step builds complete graph over the keyword nodes by either single-source shortest path or breadth-first algorithms and then finds minimum spanning trees to find approximate Steiner tree.

3.3.2.1 Finding Trees Given schema graph G and set of keyword nodes $N \subseteq V(G)$, FINDTREES algorithm (Algorithm 4) finds all trees that are possible answers to the keyword query in an exhaustive fashion. Since G might be a disconnected graph, keyword nodes might be located in different connected components and thus not all trees contain all keyword nodes. In general, $V(R_i) \subseteq N$, where R_i is one of the resulting trees and $V(R_i)$ is the set of nodes in that tree.

The main idea behind FINDTREES algorithm is similar to backwards expansion/distance network heuristics in that it starts to build complete graphs between keyword nodes. However, instead of doing it for all keyword nodes, it iterates over the keyword nodes array (using recursive FINDTREESBETWEENONEANDREST, (Algorithm 5)), and takes two nodes into consideration at a time. The algorithm searches for all paths between two vertices from the keyword nodes N. If there are no paths then one of the nodes is pushed into T array, then next node from the N is taken and the search for all paths is repeated for the new pair of nodes. If there are any paths between the pair of nodes, then those paths merge into resulting trees using MERGEPATHSTOTREES routing (Algorithm 6) and next node is taken from the N to repeat the search for all paths. When N is exhausted, then N is assigned with the nodes from T array (nodes that were not included in any tree because they are in different connected component). The algorithm repeats until both N and T are exhausted. All found trees are then weighted and sorted based on their costs.

Algorithm 4: FINDTREES(G, N). **I**nput: Schema graph G, list of keyword nodes N. **O**utput: cost based ranked list of trees.

```
1: $R \leftarrow$ empty list
2: **while** LENGTH($N$) > 0 **do**
3:     $h$ = $N$.POP()
4:     ($T$, $D$) = FINDTREESBETWEENONEANDREST(G, $h$, $N$, [], [])
5:     $R$.APPENDALL($D$)
6:     $N$ = $T$
7: **end while**
8: $R$ = CALCULATECOSTANDRANK($R$)
9: return $R$
```

Algorithm 5: FINDTREESBETWEENONEANDREST(*G, h, N, T, D*). **I**nput: Schema Graph *G*, one keyword node *h*, list of keyword nodes *N* except node *h,* accumulating parameter *T* for keyword nodes that were not included in any tree*,* accumulating parameter for found trees *D*. **O**utput: *T* and *D*.

```
1:  if LENGTH(N) = 0 then
2:      return (T, D)
3:  end if
4:  h2 = N.POP()
5:  P = ALLPATHS(h, h2)
6:  if LENGTH(P) = 0 then
7:      T.APPENDIFNOTCONTAINED(h2)
8:      return FINDTREESBETWEENONEANDREST(G, h, N, T, D)
9:  end if
10: D = MERGEPATHSTOTREES(P, D)
11: return FINDTREESBETWEENONEANDREST(G, h2, N, T, D)
```

Algorithm 6: MERGEPATHSTOTREES(*P, D*). **I**nput: Newly found paths *P*, paths found before *D*. **O**utput: list of trees created from pairwise merging paths in *P* with trees in *D*.

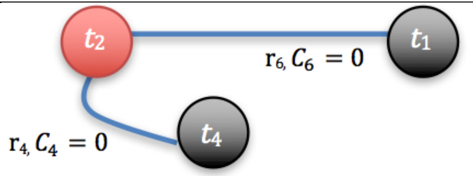
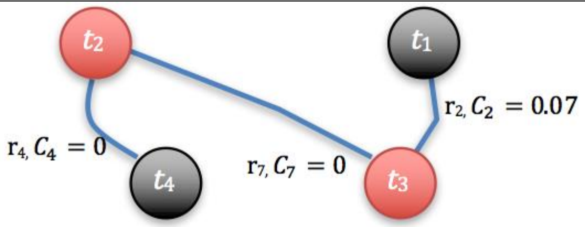
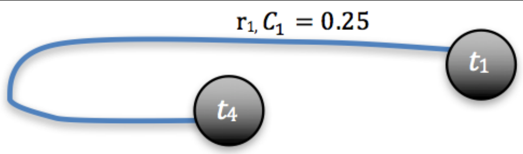
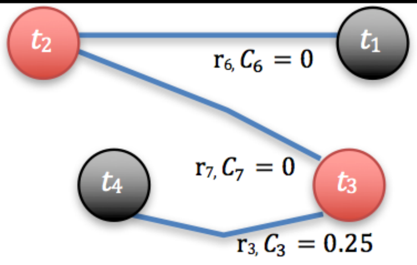
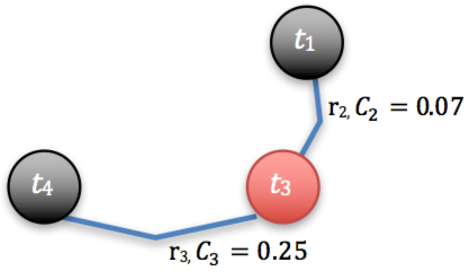
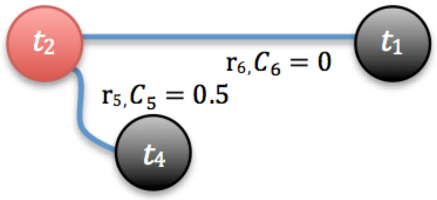
```
1:  R ← empty list
2:  if LENGTH(D) = 0 then
3:      R = D
4:  else
5:      for $\forall Pi \in P, \forall\ Di \in D$ do
6:          R.APPENDIFNOTCONTAINED($Pi \cup Di$)
7:      end for
8:  end if
9:  return R
```

In other words, the FINDTREES algorithm implicitly looks for connected components of the schema graph that contain keyword nodes and "grows" trees from all paths between every pair of keyword nodes in each connected component.

For the example keywords from Section 3.3.1 and the schema graph from Section 3.3.2, FINDTREES algorithm would find 6 trees (because of the example graph, all trees are actually paths, but in general the result of the FINDTREES algorithm is a list of trees) shown in the Table 1 sorted by the cost form lowest to highest. From the table we can see that the least-costly way to answer the keyword query is not to merge t_1 and t_4 directly, but to merge them via intermediate table t_2.

Since the schema graph can grow to a large size after many datasets are submitted, heuristics similar to SPCSH in Q [170] and DNH in [188] can be used to scale the FINDTREES algorithm. Instead of searching for all paths between pairs of nodes, we can search only m shortest (least costly) paths on line 5: in FINDTREESBETWEENONEANDREST (Algorithm 5). Efficient algorithms to solve this problem are known [189].

Table 1: Trees and their costs for the example datasets and keyword query

#	Tree	Cost
1		0
2	 t_2 t_1 $r_2, C_2 = 0.07$ $r_4, C_4 = 0$ t_4 $r_7, C_7 = 0$ t_3	0.07
3		0.25
4	 t_2 t_1 $r_6, C_6 = 0$ t_4 $r_7, C_7 = 0$ t_3 $r_3, C_3 = 0.25$	0.25
5	 t_1 $r_2, C_2 = 0.07$ t_4 t_3 $r_3, C_3 = 0.25$	0.32
6	 t_2 t_1 $r_6, C_6 = 0$ $r_5, C_5 = 0.5$ t_4	0.5

3.3.3 From Trees to Relational algebra/SQL Queries

Each tree is one item of the search result of the keyword search. However, simply returning the trees to users would expect them to integrate the datasets manually. Meanwhile, if we explore a tree more carefully, we observer that a tree can be transformed to relational algebra/SQL query.

Each node in the tree represents a data table from submitted datasets stored as a relation in a relational database in one of the *datanodes* (see Section 3.1.2); each edge represents a relationship between two tables and translates to a join operation with relationship's links as join conditions. Table ids associated with nodes and relationships ids associated with edges provide appropriate dereferencing information to construct queries by searching for the required information in the metadata tables on the metadata node.

For example, (4) shows relational algebra query constructed based on tree 1 from Table 1.

$$t_1 \bowtie_{t_1.State=t_2.State} t_2 \bowtie_{t_2.Abbreviation=t_4.State} t_4 \tag{4}$$

Relationship algebra expression (4) highlights one limitation of current approach to keyword search: even though t_1 and t_4 are related to each other via two variables (*year* and *state*), only *state* variable is used to perform the merge. To solve the problem for this particular case, we could apply selection operation (σ) as in (5), however in general additional joins need to be performed that might include some other intermediate tables. The solution for this limitation is left for future work.

$$\sigma_{t_1.Year=t_4.Year}\left(t_1 \bowtie_{t_1.State=t_2.State} t_2 \bowtie_{t_2.Abbreviation=t_4.State} t_4\right) \tag{5}$$

3.3.4 From Queries to Merged Data Tables

The final step in answering keyword search is to execute queries constructed in Section 3.3.3. Since the data tables are stored on different *datanodes*, a distributed query engine is required. First I used Linked Servers feature of Microsoft SQL Server [132] product, however it was not callable enough. Currently, I am experimenting with PrestoDB [147] distributed query engine.

Table 2 shows sample of the result of query (4) as merged table that according to the tree 1 from Table 1 joins t_1 and t_2 on $t_1.State = t_2.State$ and then with t_4 on $t_2.Abbreviation = t_4.State$. As explained above in Section 3.3.3, t_1 and t_4 merged via t_2 only using *state* variable.

Table 2: Sample of the merged table for the tree 1 from Table 1

t_1.S	t_1.Y	t_1.M	t_1.D	t_1.N	t_2.A	t_2.S	t_4.Po	t_4.Y	t_4.S*
Alabama	1888	8	enteric fever	1	AL	Alabama	1830	1900	AL
Alabama	1888	10	whooping cough	1	AL	Alabama	1830	1900	AL
Alabama	1888	10	enteric fever	1	AL	Alabama	1830	1900	AL
Alabama	1890	9	diphtheria	1	AL	Alabama	1830	1900	AL
Alabama	1891	3	phthisis pulmonalis	12	AL	Alabama	1830	1900	AL

Since datasets can be merged via one or more other datasets, search result should include provenance information [32][91][135] of the merged table, so that user can fully understand how the table was constructed. Thus, in addition to the merged data table, query result also includes information on all relationships and intermediate datasets that were used to construct the merged table.

Most of related works on keyword search assumed query-independent costs for query trees such as number of edges, attribute similarity, etc., when users may need costs specific to the context of the query. Preferences for sources may depend on whether users are posing “what-if” types of exploratory queries or refining previous answers. I suggest to performs *CPR-ranking*: the data merge trees are ranked based on their *coverage*, *precision* and *reliability* that are derived from relationships confidence, data overlapping values, path length, data and user reliability. As it was mentioned in Section 3.2.3.3, Schema graph cost model depends on the weight vector $\bar{\boldsymbol{w}}$. In contrast to all the previous work, I propose this vector to be configured by users. By modifying weights for specific features, user can control CPR ranking.

4.0 SYSTEM IMPLEMENTATION: COL*FUSION

For the last two years I have been working on the implementation of the infrastructure described in Section 3.0in the Col*Fusion (Collaborative data Fusion) system. I have been also supervising a group of master students who were helping me with the implementation.

Col*Fusion can be accessed via this URL: http://colfusion.exp.sis.pitt.edu/colfusion/, however the registration is currently not open to the public and requires verification code that users can get by invitation.

In this chapter I describe Col*Fusion architecture, current state of its implementation and main functionality in Section 4.1. In Section 4.2 I enumerate current and/or future implementation tasks to improve Col*Fusion. I then, in Section 4.3, show how Col*Fusion has been utilized in Collaborative for Historical Information and Analysis (CHIA) and what lessons we learned from it.

4.1 ARCHITECTURE, IMPLEMENTATION, OPERATIONS

Col*Fusion is in the active development (currently it is around 60K lines of code). Figure 18 shows major components of the Col*Fusion architecture (gray boxes represent unfinished modules/functionality). Col*Fusion is designed in the modular way such that it is easy to replace specific modules if needed as well as to distribute Col*Fusion execution among cluster of machines. The Col*Fusion hardware and software architecture is designed to enable effective data integration and analysis through crowdsourcing at the *interactive rates* that people expect of web-based resources. Achieving that level of performance for high-volume, high-variety data requires coupling scalable clusters for relational and non-SQL databases, web interfaces, and many software tools with high-performance, purpose-built computational resources for complex analytics. Large-scale data must move between those resources efficiently, reliably, and transparently, including optimizations to reduce or eliminate latency and to maximize use of finite network bandwidth.

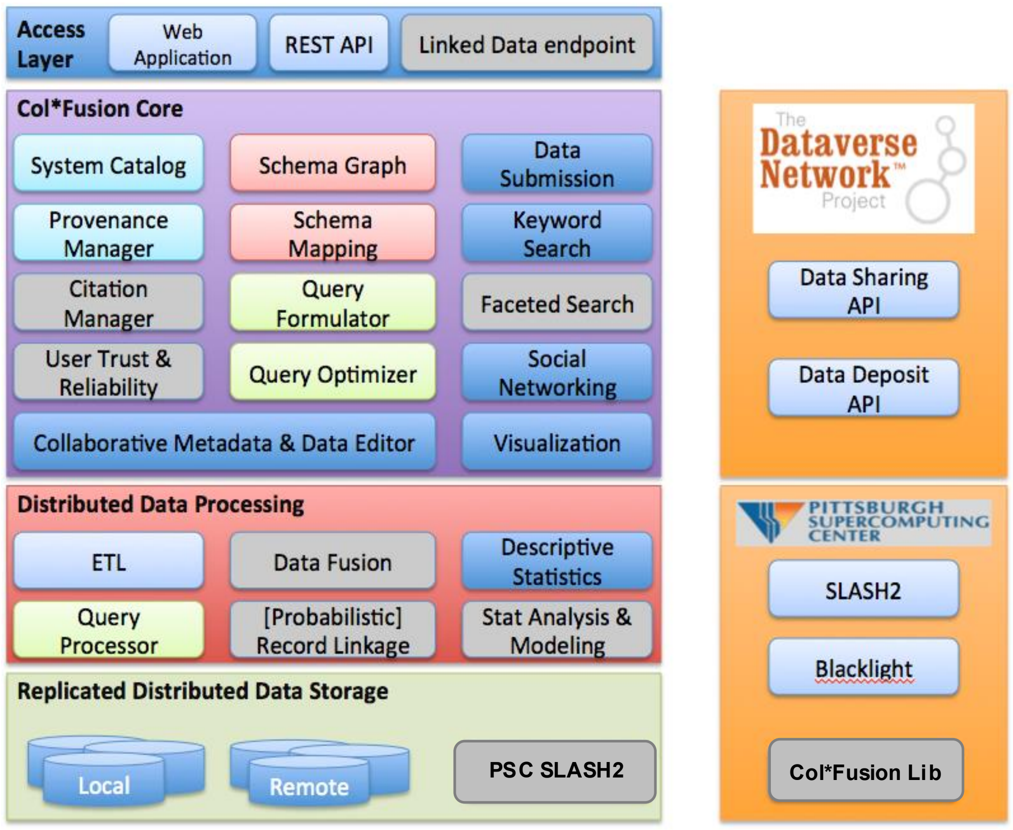


Figure 18: Col*Fusion Architecture (gray boxes represent unfinished modules/functionality)

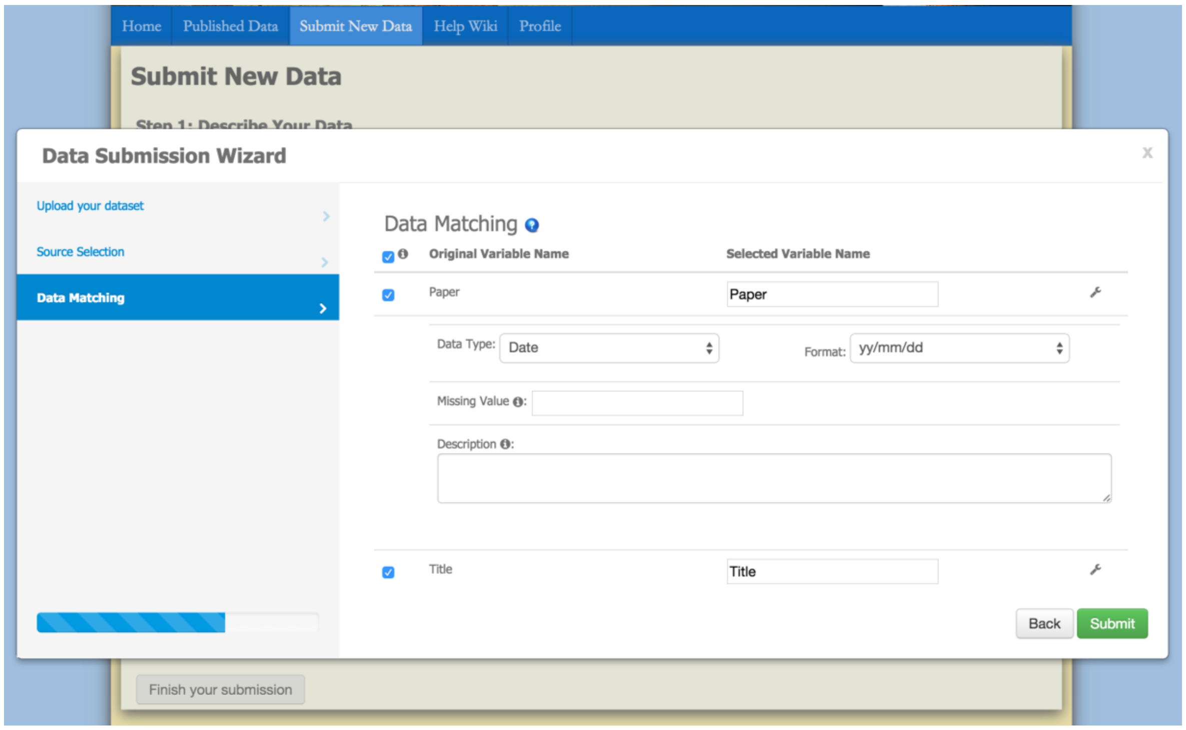
Col*Fusion architecture consists of four internal components: (1) *Access Layer* component that provides several interfaces so that users and third party tools can interact with Col*Fusion, (2) *Col*Fusion Core* component that is responsible for the business logic and metadata management, (3) *Distributed Data Processing* component that handles large scale data processing on distributed set of commodity machines, (4) *Replicated Distributed Data Storage* component that is responsible for storing all datasets; and two external components: (1) *The Dataverse Network* [106][43] project developed at Harvard that is used as data archive, and (2) *Pittsburgh Supercomputing Center*[1] facilities used as the large scale high-performance computing resource. I have successfully collaborated with PSC staff to bridge Col*Fusion and PSC resources. On Col*Fusion site we have setup PSC SLASH 2 [164] distributed replicated storage system. On PSC site we have setup all required data store systems (relational and graph) as well as agreed on workflow for Col*Fusion Lib that will be running web server to accept Col*Fusion commands and execute large-scale computations on PSC computing framework. In what follows, I briefly describe Col*Fusion main functionality and refer to corresponding modules as well as present screenshots of the user interface where appropriate.

4.1.1 Data and Metadata Submission

Col*Fusion system does not require either special software installation or prior knowledge of a specific data management systems. It supports a simple data submission protocol implemented via lightweight intuitive web interface shown on Figure 19. Col*Fusion also implements a RESTful [67] API for direct access to Col*Fusion functionalities.

[1] http://www.psc.edu/

Data submission module allows users to submit data from heterogeneous sources in various formats such as Excel, CSV files, and database dump files, as well as remotely connected databases. The number of file formats can be expanded by Col*Fusion users. Col*Fusion uses Pentaho Data Integration [35] (aka Kettle) on the back end for extracting, transforming and loading (ETL) data into Col*Fusion repository. Kettle is an open-source software that allows users to specify ETL tasks via intuitive, graphical, drag and drop design environment and save it as a transformation file. Kettle supports large number of data sources including leading Hadoop distributions, NoSQL databases, and other big data stores. Col*Fusion users can create Kettle transformations and share them with other users to handle particular file format. Therefore, most users do not need to do any preparatory work to submit their datasets into Col*Fusion, which makes it easy to use.



Original Variable Name	Selected Variable Name
Paper	Paper
Title	Title

Data Type: Date
Format: yy/mm/dd
Missing Value:
Description:

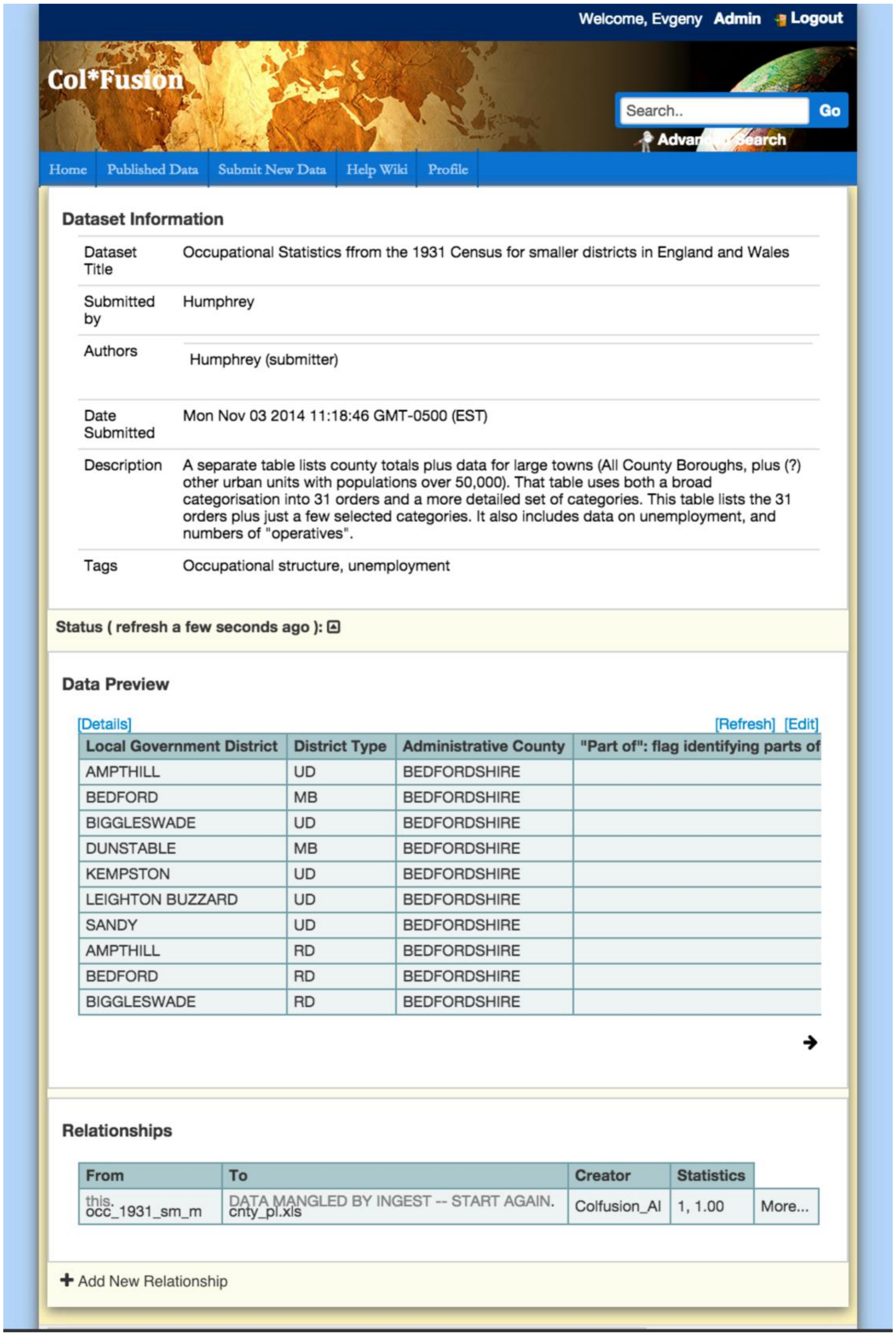
Figure 19: Data submission page

During data submission Col*Fusion tries to collect as much metadata as possible on both dataset as well as variable levels. The metadata is either retrieved from the data file (e.g., variables names, data types, etc.) or entered by the user (e.g., title, description, tags, category, variables format and measuring units, etc.).

The System Catalog persistently stores the metadata and allocates a data storage server. Col*Fusion then triggers a background ETL process to extract and transform data from user provided source and load it into Col*Fusion *datanode*. To guarantee durable storage, I have started to utilize distributed replicated storage system backed by Pittsburgh Super Computing (PSC) center's SLASH2 system [164]. SLASH2 supports automated file replication and migration operations which Col*Fusion can leverage to maintain working copies of data near appropriate computational resources.

4.1.2 Data Access and Export

For each submitted dataset, Col*Fusion creates a dedicated identifiable page (I call it a Story page) on which users can view metadata and data in a tabular paged format (Figure 20); discuss the dataset in form of dataset, row, column or cell level comments; visualize data with interactive pie, column, map or motion charts (Figure 21); download data in Excel, CSV, JSON, HTML table formats regardless of the original file format. As it was said above, Col*Fusion also provides RESTful API that provide programmatic access to metadata and data.


Welcome, Evgeny Admin Logout
Col*Fusion
Search.. Go
Advanced Search
Home Published Data Submit New Data Help Wiki Profile

Dataset Information

Dataset Title	Occupational Statistics ffrom the 1931 Census for smaller districts in England and Wales
Submitted by	Humphrey
Authors	Humphrey (submitter)
Date Submitted	Mon Nov 03 2014 11:18:46 GMT-0500 (EST)
Description	A separate table lists county totals plus data for large towns (All County Boroughs, plus (?) other urban units with populations over 50,000). That table uses both a broad categorisation into 31 orders and a more detailed set of categories. This table lists the 31 orders plus just a few selected categories. It also includes data on unemployment, and numbers of "operatives".
Tags	Occupational structure, unemployment

Status (refresh a few seconds ago):

Data Preview

[Details] [Refresh] [Edit]

Local Government District	District Type	Administrative County	"Part of": flag identifying parts of
AMPTHILL	UD	BEDFORDSHIRE	
BEDFORD	MB	BEDFORDSHIRE	
BIGGLESWADE	UD	BEDFORDSHIRE	
DUNSTABLE	MB	BEDFORDSHIRE	
KEMPSTON	UD	BEDFORDSHIRE	
LEIGHTON BUZZARD	UD	BEDFORDSHIRE	
SANDY	UD	BEDFORDSHIRE	
AMPTHILL	RD	BEDFORDSHIRE	
BEDFORD	RD	BEDFORDSHIRE	
BIGGLESWADE	RD	BEDFORDSHIRE	

Relationships

From	To	Creator	Statistics	
this. occ_1931_sm_m	DATA MANGLED BY INGEST -- START AGAIN. cnty_pl.xls	Colfusion_AI	1, 1.00	More...

+ Add New Relationship

Figure 20: Story page: Data Preview

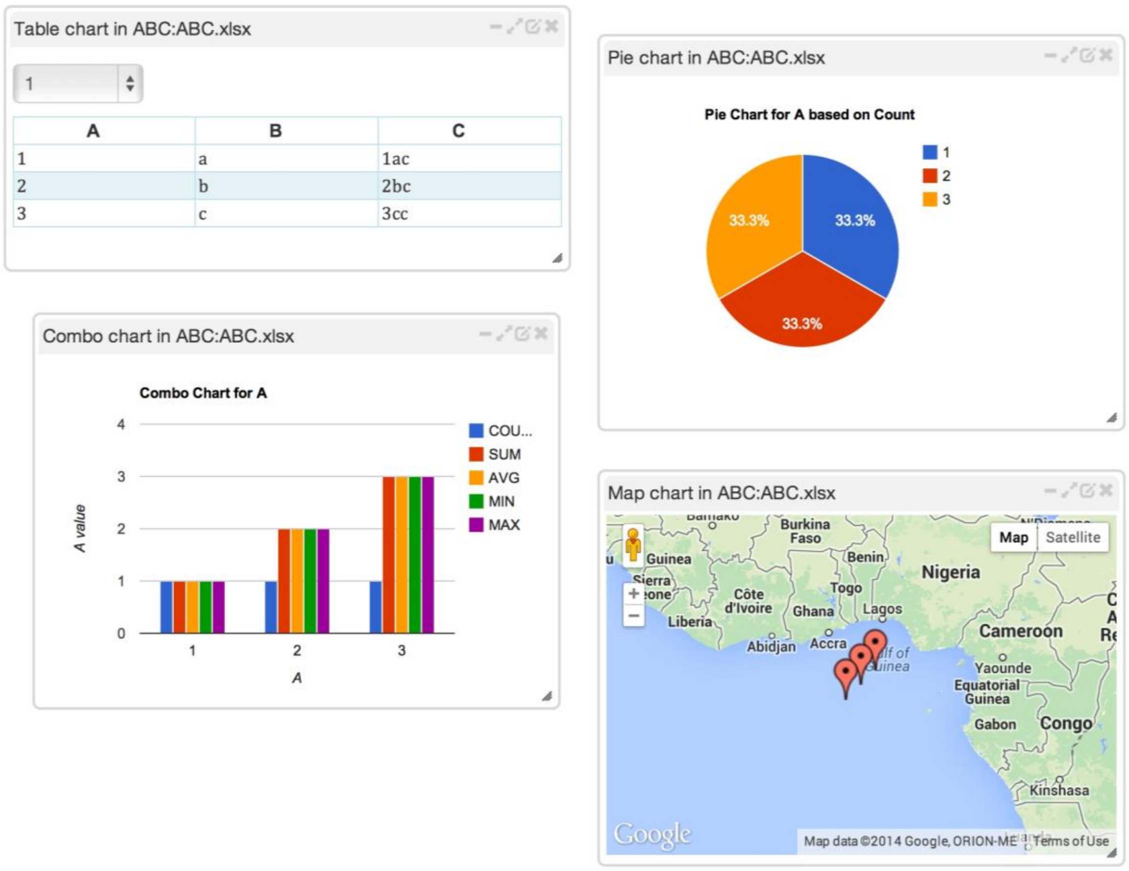


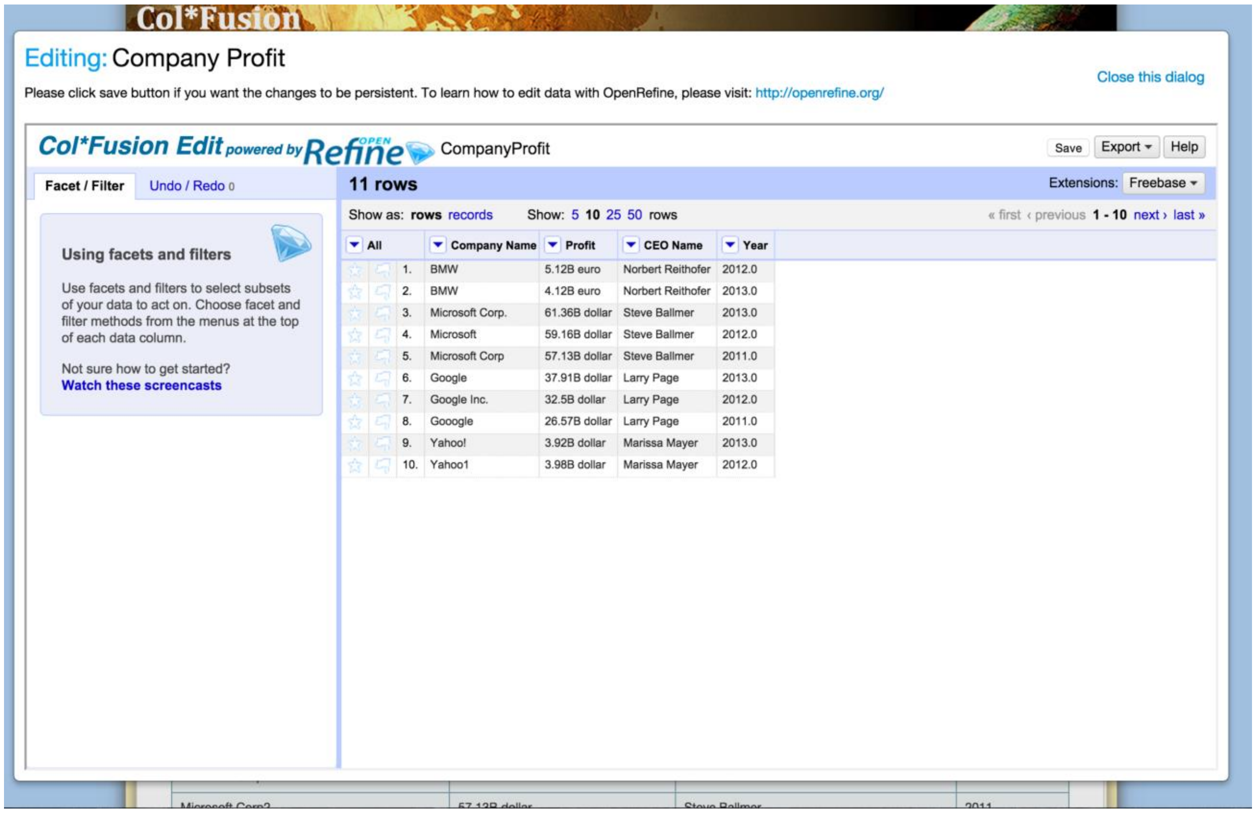
Table chart in ABC:ABC.xlsx

A	B	C
1	a	1ac
2	b	2bc
3	c	3cc

Figure 21: Visualization

4.1.3 Collaborative Metadata and Data Editing

All metadata and data can be collectively edited by registered Col*Fusion users anytime after submission. Data editing is supported by an integrated OpenRefine editor [181] that provides basic and advanced cell and column edit and transformation functionalities (Figure 22). Provenance Manager keeps track of all changes and performs automatic versioning that allows to rollback any undesired changes.



Editing: Company Profit

Please click save button if you want the changes to be persistent. To learn how to edit data with OpenRefine, please visit: http://openrefine.org/

Close this dialog

Col*Fusion Edit powered by OpenRefine CompanyProfit

Save Export Help

Facet / Filter Undo / Redo 0

Using facets and filters

Use facets and filters to select subsets of your data to act on. Choose facet and filter methods from the menus at the top of each data column.

Not sure how to get started?
Watch these screencasts

11 rows

Extensions: Freebase

Show as: **rows** records Show: 5 **10** 25 50 rows

« first ‹ previous **1 - 10** next › last »

All	Company Name	Profit	CEO Name	Year
1.	BMW	5.12B euro	Norbert Reithofer	2012.0
2.	BMW	4.12B euro	Norbert Reithofer	2013.0
3.	Microsoft Corp.	61.36B dollar	Steve Ballmer	2013.0
4.	Microsoft	59.16B dollar	Steve Ballmer	2012.0
5.	Microsoft Corp	57.13B dollar	Steve Ballmer	2011.0
6.	Google	37.91B dollar	Larry Page	2013.0
7.	Google Inc.	32.5B dollar	Larry Page	2012.0
8.	Gooogle	26.57B dollar	Larry Page	2011.0
9.	Yahoo!	3.92B dollar	Marissa Mayer	2013.0
10.	Yahoo1	3.98B dollar	Marissa Mayer	2012.0

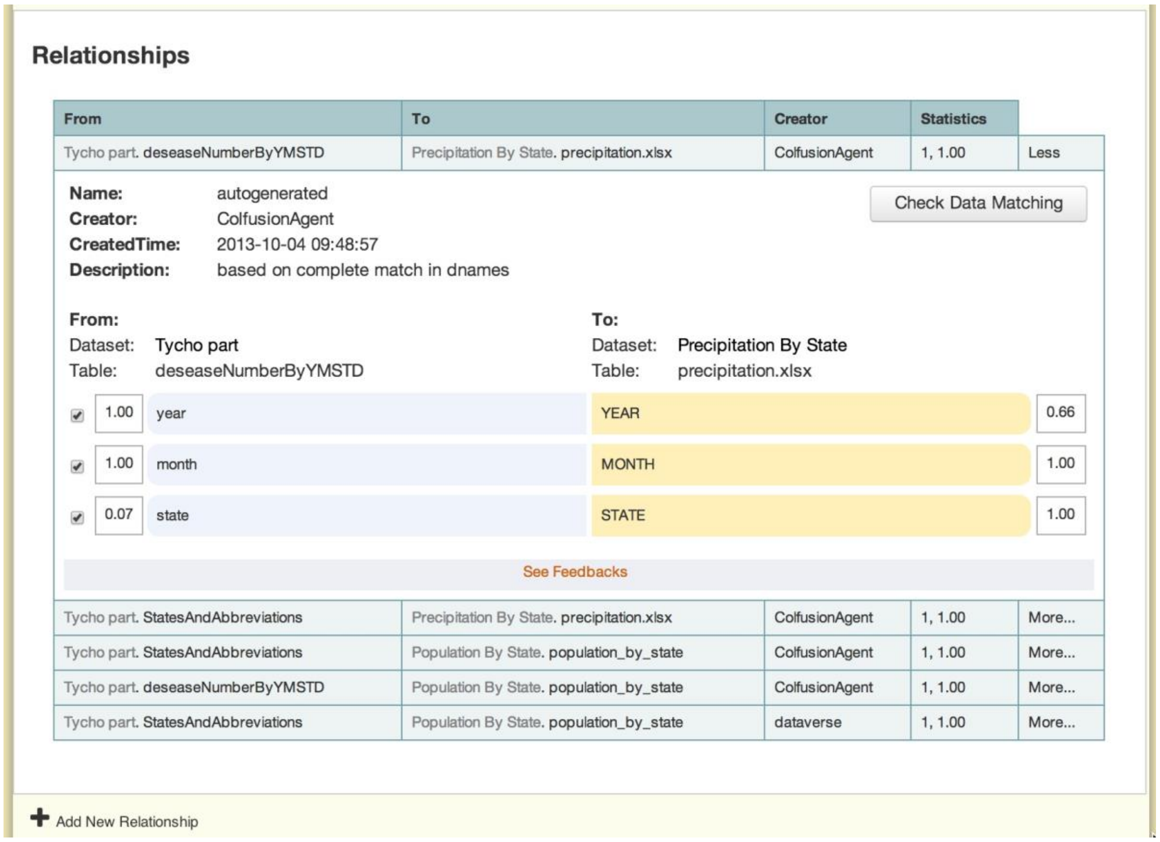
Figure 22: Interface to edit data table via OpenRefine

4.1.4 Information Linkage

Once the user confirms data submission, Col*Fusion performs information linkage that includes relationships discovery between the newly submitted dataset and existing datasets in the Col*Fusion repository as it was described in Section 3.2. Currently Col*Fusion establishes a relationship between datasets based on linguistic similarity over variables metadata in those datasets. Many different schema matchers that incorporate multiple features and use advanced machine learning algorithms have been proposed in recent years [149], with one of the most sophisticated being COMA++ [150]. Extendable architecture of Col*Fusion allows to easily utilize those tools.

Story page for each dataset includes Relationships section that is shown in Figure 23. Relationships are listed in the table view and each row in the table can be expanded to see relationship's metadata (name, description, creator of the relationship, and time and date when it was created) and links. Each link has two *DO* values at its ends.

Col*Fusion combines automatic relationship discovery with the power of crowdsourcing techniques. Col*Fusion users can provide feedback on automatically generated relationships in terms of confidence values that reflect users' belief that relationships hold. Also Col*Fusion users can manually create a relationship if Col*Fusion fails to identify one (Figure 24).



Relationships

From	To	Creator	Statistics	
Tycho part. deseaseNumberByYMSTD	Precipitation By State. precipitation.xlsx	ColfusionAgent	1, 1.00	Less

Name: autogenerated
Creator: ColfusionAgent
CreatedTime: 2013-10-04 09:48:57
Description: based on complete match in dnames

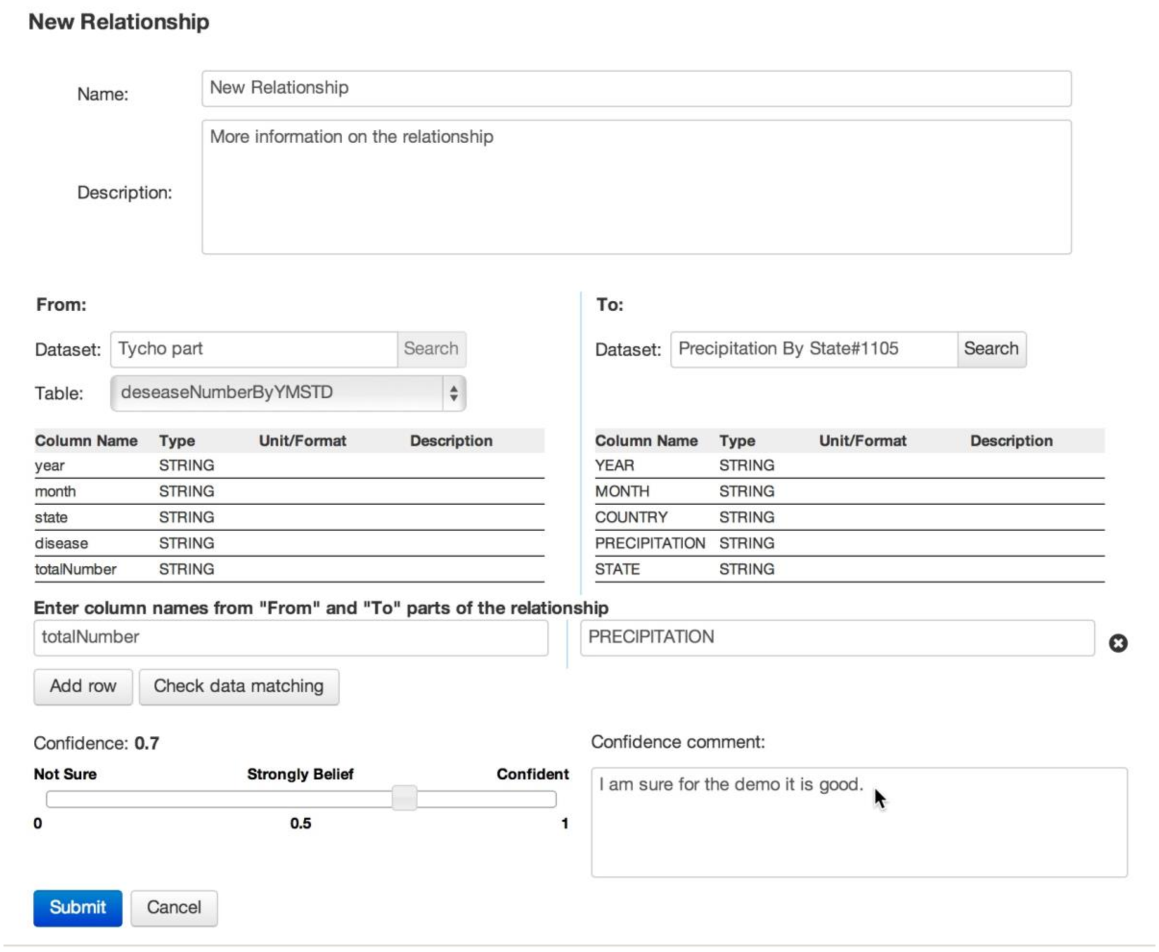
From:
Dataset: Tycho part
Table: deseaseNumberByYMSTD

To:
Dataset: Precipitation By State
Table: precipitation.xlsx

1.00	year	YEAR	0.66
1.00	month	MONTH	1.00
0.07	state	STATE	1.00

From	To	Creator	Statistics	
Tycho part. StatesAndAbbreviations	Precipitation By State. precipitation.xlsx	ColfusionAgent	1, 1.00	More...
Tycho part. StatesAndAbbreviations	Population By State. population_by_state	ColfusionAgent	1, 1.00	More...
Tycho part. deseaseNumberByYMSTD	Population By State. population_by_state	ColfusionAgent	1, 1.00	More...
Tycho part. StatesAndAbbreviations	Population By State. population_by_state	dataverse	1, 1.00	More...

Figure 23: Relationship table



New Relationship

Name:	New Relationship
Description:	More information on the relationship

From:

Dataset: Tycho part

Table: deseaseNumberByYMSTD

Column Name	Type	Unit/Format	Description
year	STRING		
month	STRING		
state	STRING		
disease	STRING		
totalNumber	STRING		

To:

Dataset: Precipitation By State#1105

Column Name	Type	Unit/Format	Description
YEAR	STRING		
MONTH	STRING		
COUNTRY	STRING		
PRECIPITATION	STRING		
STATE	STRING		

Enter column names from "From" and "To" parts of the relationship

totalNumber — PRECIPITATION

Confidence: 0.7

Confidence comment: I am sure for the demo it is good.

Figure 24: Add new relationship

Col*Fusion doesn’t have a predefined target schema where data from heterogeneous data sources are supposed to be loaded to as in traditional data warehousing approach. Instead it maintains a global *Schema Graph* – an undirected graph where nodes represent datasets and edges represent relationships. Neo4j graph database [136] (the leading graph database) is utilized to store and traverse Schema Graph.

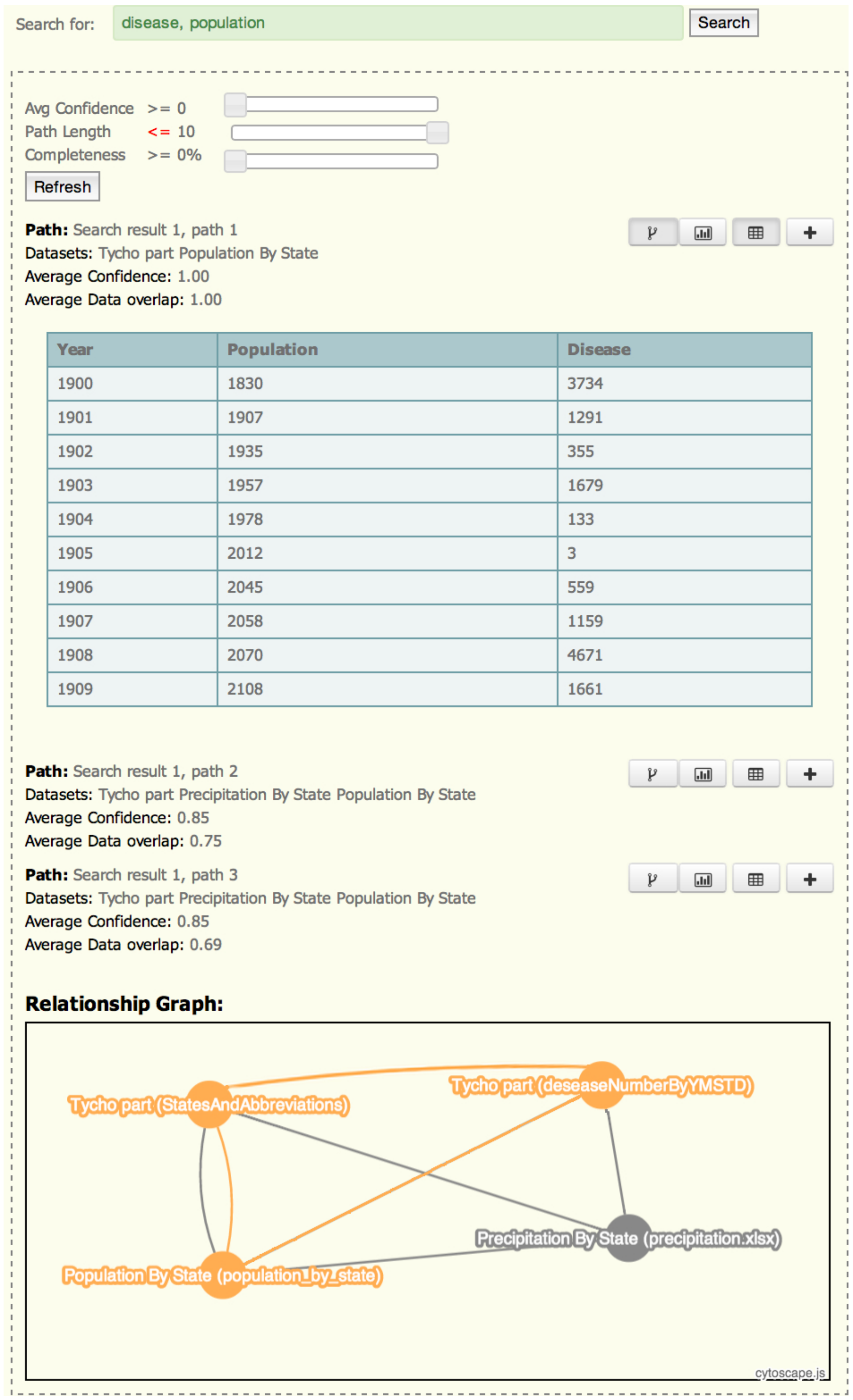
Col*Fusion also provides interactive visualization of the Schema Graph on the Story page that allows users to see how their (or any) dataset is connected to other datasets and provide a way to discover new, possibly interesting, data that will lead to new hypothesis to test.

4.1.5 Search and Exploration

Col*Fusion provides full text search throughout datasets metadata such as title, description and tags. It also provides keyword search that is quite different from the keyword search functionality in all existing data repositories that only search through datasets metadata and return a list of datasets that might contain data that user is interested in.

Figure 25 shows Col*Fusion interface and result of the keyword search. When user posts a keyword query, Col*Fusion first finds all datasets (nodes in the Schema Graph) that contain the keywords, then traverses the schema graph to find all trees between those nodes, passes each three to the Query Formulator that converts trees to SQL queries, and then finally let Query Processor to execute those queries. The result of the search is not just a list of datasets that might have data that user is interested in, but rather a *merged* dataset or a list of merged datasets if there are several possible paths to answer the query. Col*Fusion performs *CPR-ranking*: the data merge paths are ranked based on their *coverage*, *precision* and *reliability* that are derived from relationships confidence, data overlapping values, path length, data and user reliability.

The keyword search allows users to search for data transparently throughout all submitted datasets without a prior knowledge of any query language or the relationships between the datasets.


Search for: disease, population Search

Avg Confidence >= 0
Path Length <= 10
Completeness >= 0%

Refresh

Path: Search result 1, path 1
Datasets: Tycho part Population By State
Average Confidence: 1.00
Average Data overlap: 1.00

Year	Population	Disease
1900	1830	3734
1901	1907	1291
1902	1935	355
1903	1957	1679
1904	1978	133
1905	2012	3
1906	2045	559
1907	2058	1159
1908	2070	4671
1909	2108	1661

Path: Search result 1, path 2
Datasets: Tycho part Precipitation By State Population By State
Average Confidence: 0.85
Average Data overlap: 0.75

Path: Search result 1, path 3
Datasets: Tycho part Precipitation By State Population By State
Average Confidence: 0.85
Average Data overlap: 0.69

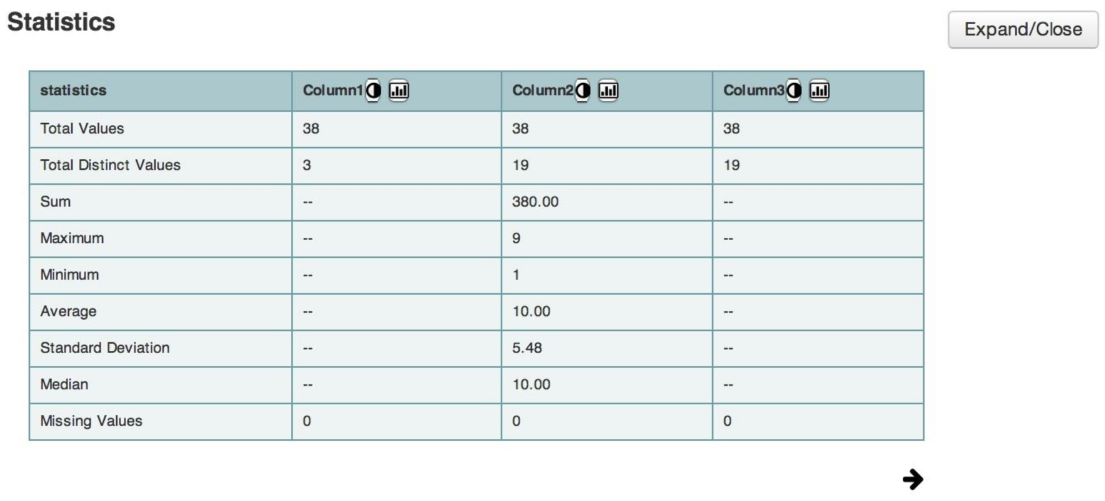
Relationship Graph:

Tycho part (StatesAndAbbreviations)
Tycho part (deseaseNumberByYMSTD)
Precipitation By State (precipitation.xlsx)
Population By State (population_by_state)
cytoscape.js

Figure 25: Keyword search interface

4.1.6 Descriptive Statistics and Data Analysis

For each submitted dataset Col*Fusion automatically calculates some descriptive statistics, such as min, max, mean, mode, median, count, standard deviation as well as correlation matrix that contains pairwise correlation values between all variables (Figure 26). The descriptive statistics can also be visualized on pie, bar, line and scatter plot charts.



statistics	Column1	Column2	Column3
Total Values	38	38	38
Total Distinct Values	3	19	19
Sum	--	380.00	--
Maximum	--	9	--
Minimum	--	1	--
Average	--	10.00	--
Standard Deviation	--	5.48	--
Median	--	10.00	--
Missing Values	0	0	0

Figure 26: Descriptive statistics table that Col*Fusion automatically creates for each dataset

4.2 ACTIVE AND FUTURE IMPLEMENTATION TASKS

The above mentioned functionality has been implemented and deployed. In addition to being a challenging research project, Col*Fusion has also been used as a productive educational test-bed. Over the course of the last two years about 60 master students have acquired hands-on practical skills and knowledge in building complex large-scale data intensive system either in the form of individual study or as a class term project. In this section I discuss a number of active and future tasks that I plan to be implemented in Col*Fusion.

First of all, to utilize functionality of established data repository, such as data archival and preservation, citation, and metadata export in Dublin Core [185][59] and DDI [155][47] formats, I plan to integrate Col*Fusion with The Dataverse Network [106] over the Dataverse Sharing and Deposit APIs.

As it was mentioned in Section 4.1.5, for each dataset Provenance Manager maintains provenance information. Dataset's provenance can be visualized as provenance graph in Open Provenance Model (OPM) format [134] that can be either viewed as interactive graph (Figure 27) visualization or downloaded in XML format.

Since Col*Fusion actually processes the submitted data files and extracts and loads data into Col*Fusion data store, with the proper interface it is possible to utilize Col*Fusion as cloud data analytical platform where researcher would be able to run complex data analysis tasks. IPython Notebook [144] is a perfect tool for "… web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document".

I plan to integrate IPython Notebook into Col*Fusion. Thus in the future, in addition to the integrated data repository, I envision Col*Fusion to be a virtual collaborative research environment where researcher can communicate their ideas, run analysis on the integrated data, build and share interactive visualizations, even write and publish online interactive research papers with data and analysis code integrated into them.

Finally, to connect Col*Fusion integrated data to the Linked Data cloud [22], I plan to implement Col*Fusion Linked Data endpoint that will allow Col*Fusion data to be presented and queried according to linked data principles. For Linked Data endpoint I am planning to use Apache Marmotta [9] with Linked Data Fragments [180] that are the state of the art technology to large scale linked data querying.

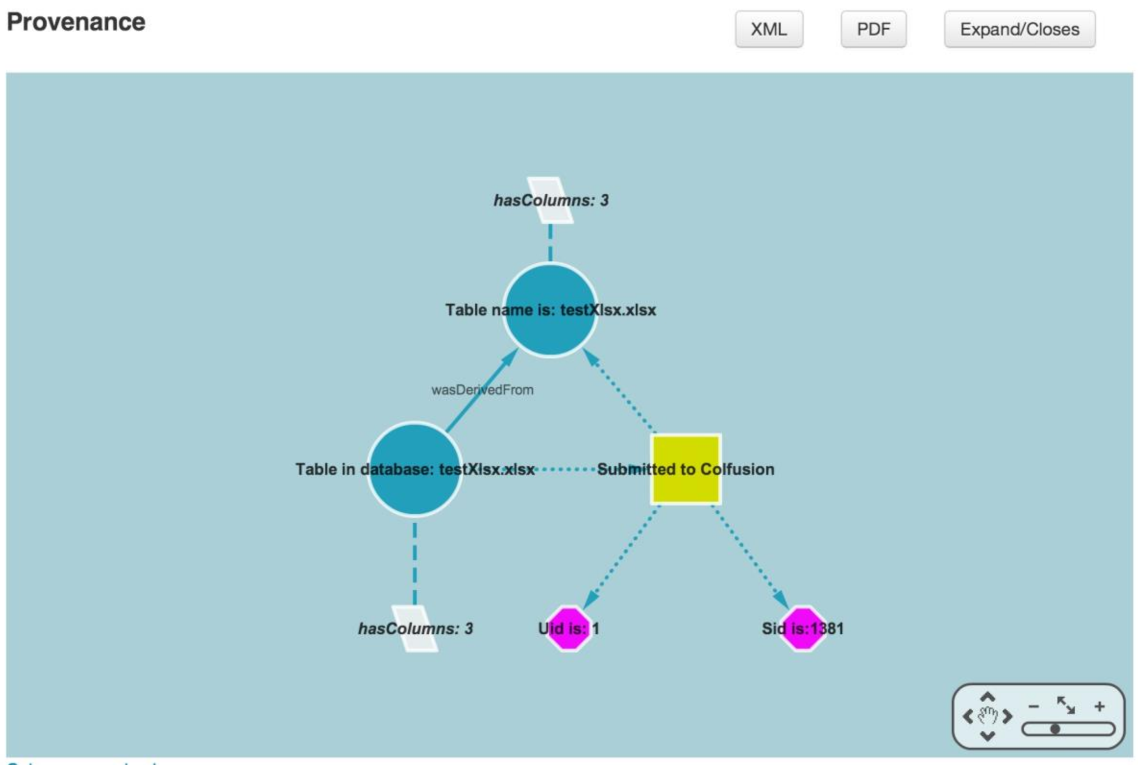


Figure 27: Provenance graph

Even though Col*Fusion is still in development, we have opened it to a group of users for real life use cases. This collaboration is explained in the next section.

4.3 REAL-LIFE DATA-INTENSIVE USAGE

To apply Col*Fusion in real-life settings, we worked together with the Collaborative for Historical Information and Analysis (CHIA) (http://www.chia.pitt.edu/) that is headquartered at World History Center, University of Pittsburgh. CHIA currently involves nine different research groups throughout the U.S. and Europe; it aims to create a major repository of consolidated global historical data from the past several centuries. CHIA mission includes following statement:

"The long-term purpose of CHIA—in the time frame of a decade or more—is to facilitate the creation and maintenance of historical a world-historical archive including data from local to global levels, from short term to long term, linking variables on many areas of human experience. The resultant summation of human experience can reveal the varying patterns and dynamics of social change. While past social, economic, and cultural dynamics may not carry automatically into the future, they should not be neglected in our attempts to make plans and form policy."

CHIA members have used Col*Fusion to contribute more than 70 datasets on various topics and provided a constructive and positive feedback. Table 3 shows the inventory of the datasets submitted by CHIA members.

Overall the collaboration with CHIA was successful and fruitful. Col*Fusion became tightly integrated into CHIA infrastructure that was reflected in the book by CHIA director Dr. Patrick Manning [126] as well as in [191]. We have presented Col*Fusion on the CHIA workshop in May 2013 and World Historical Gazetteer workshop organized by CHIA in September 2014.

Table 3: Col*Fusion Dataset Inventory from CHIA participants

File / Data Set Name	Distr.	start date	end date	Spatial continental	Spatial national	spatial.3	Topic	variables
Codebooks								
Geo Names	Pitt	-		-			place names	multi level spatial labels
Data Sets								
Deaths per thousand population in Java, Indonesia between 1916 and 1920	MSU	1916	1920	Asia	Indonesia	Java	Death and Disease	Deaths, Disease, Influenza, Pandemic
Hospital beds data of Punjab between 1907 and 1919	MSU	1907	1919	Asia	India	Punjab	Death and Disease	Disease, Hospital beds, gender
Labour Relations The Netherlands 1900	IISH	1900	1900	Europe	Netherlands		Demographic	Labor
Age Structure of Population of Punjab, 1901 to 1931	MSU	1901	1931	Asia	India	Punjab	Demographic	Age
Literacy in Punjab, 1901 to 1931	MSU	1901	1931	Asia	India	Punjab	Demographic	Literacy
Religion in Punjab, 1901 to 1931	MSU	1901	1931	Asia	India	Punjab	Demographic	Religion
Correlates of War data	Pitt	1816	2010	Global			Historical	War, Initiator, Intervener, Outcome, Deaths
Population / CLIO World Tables	BU	1335	2006	Global			Population	Population
Government / CLIO World Tables (Countries Only)	BU	1826	2008	Global			Other	Expenditure, Debt, Military, Revenue, Currency
African Population Totals, 1850-1960	Pitt	1850	1960	Africa			Population	Population
UN Department of Economic and Social Affairs Population Data	Pitt	1950	2010	Global			Population	Population
Global Historical Climatology Network (GHCN-Monthly) historical precipitation data	Pitt	1951	1975	Global			Natural Phenomena	Precipitation

Table 3 (continued)

File / Data Set Name	Distr.	start date	end date	Spatial continental	Spatial national	spatial.3	Topic	variables
Population of Rajputana, 1891 to 1941	MSU	1891	1941	Asia	India	Rajputana	Population	Population
Population trajectory of India, 1891 to 1941	MSU	1891	1941	Asia	India		Population	Population
Population trajectory of Japan, 1903 to 1929	MSU	1903	1929	Asia	Japan		Population	Population
Population trajectory of Java, Indonesia, 1880 to 1930	MSU	1880	1930	Asia	Indonesia		Population	Population
Population trajectory of Srilanka (Ceylon) from 1891 to 1946	MSU	1891	1946	Asia	Srilanka		Population	Population
Monthly mortality data of Assam from 1916 to 1921	MSU	1916	1921	Asia	India	Assam	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Bengal from 1916 to 1921	MSU	1916	1921	Asia	India	Bengal	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Bihar from 1916 to 1921	MSU	1916	1921	Asia	India	Bihar	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Bombay from 1916 to 1921	MSU	1916	1921	Asia	India	Bombay	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Central Province from 1916 to 1921	MSU	1916	1921	Asia	India	Central Province	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Madras from 1916 to 1921	MSU	1916	1921	Asia	India	Madras	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers

Table 3 (continued)

File / Data Set Name	Distr.	start date	end date	Spatial continental	Spatial national	spatial.3	Topic	variables
Monthly mortality data of North-West Frontier from 1916 to 1921	MSU	1916	1921	Asia	India	North-West Frontier	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of United Provinces from 1916 to 1921	MSU	1916	1921	Asia	India	Uttar Pradesh	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Monthly mortality data of Punjab from 1916 to 1921	MSU	1916	1921	Asia	India	Punjab	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Assam from 1916 to 1921	MSU	1916	1921	Asia	India	Assam	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Bengal from 1916 to 1921	MSU	1916	1921	Asia	India	Bengal	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Bihar from 1916 to 1921	MSU	1916	1921	Asia	India	Bihar	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Bombay from 1916 to 1921	MSU	1916	1921	Asia	India	Bombay	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Central Province from 1916 to 1921	MSU	1916	1921	Asia	India	Central Province	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of Madras from 1916 to 1921	MSU	1916	1921	Asia	India	Madras	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of North-West Frontier from 1916 to 1921	MSU	1916	1921	Asia	India	North-West Frontier	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers

Table 3 (continued)

File / Data Set Name	Distr.	start date	end date	Spatial continental	Spatial national	spatial.3	Topic	variables
Weekly mortality data of Punjab from 1916 to 1921	MSU	1916	1921	Asia	India	Punjab	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Weekly mortality data of United Provinces from 1916 to 1921	MSU	1916	1921	Asia	India	Uttar Pradesh	Death and Disease	Deaths, Disease, Influenza, Pandemic, Fevers
Migration data of Punjab for 1911 and 1921	MSU	1911	1921	Asia	India	Punjab	Population	Population, Migration, Birth
Drug shops in Bengal between 1910 and 1940	MSU	1910	1940	Asia	India	Bengal	Commodities	Drugs, Opium, Ganja, Charas, Bhang, Majum, Cocaine
Environmental Indicators Of Air Pollution: Nox Emissions	UC-Merc	1990	2010	Global			Natural Phenomena	Air Pollution, NOx Emissions, Environment
Consumption of Ozone Depleting Substances and CFCs	UC-Merc	2002	2008	Global			Natural Phenomena	Ozone, CFCs, Pollution, Environment
Occupational statistics for the local government districts of England and Wales in 1921	Portsmouth	1921	1921	Europe	Great Britian	England, Wales	Demographic	Occupation
Redistricted age/gender structure data for the modern local government districts of Great Britian 1851 to 2001	Portsmouth	1851	2001	Europe	Great Britian		Demographic	Age, Gender
Opium Import, Export, and Price Data for New York Chamber of Commerce 1870 - 1918	WHD	1870	1918	North America	USA	New York	Commodities	Opium, Import, Export, Price

Table 3 (continued)

File / Data Set Name	**Distr.**	**start date**	**end date**	**Spatial continental**	**Spatial national**	**spatial.3**	**Topic**	**variables**
Andes Termperate-Mediterranean Transition 657 Year PDSI Reconstruction	UC-Merc	1350	2014	South America	Peru		Natural Phenomena	Tree ring, Drought, Andes, Climate, Temperature
Global Temperature, RSL, Ice Volume 1,000,000	UC-Merc	n/a	n/a	Global			Natural Phenomena	Temperature, Ice, Climate
20,000 Year Borehole Surface Temperature Reconstruction	UC-Merc	n/a	n/a	Global			Natural Phenomena	Temperature, Ice, Climate
North Atlantic, European and Mediterranean Gridded SLP Reconstruction	UC-Merc	1750	1850				Natural Phenomena	Climate
Longxi China 1000 Year Decadal Hydrological Indices AD 960-1990	UC-Merc	960	1990	Asia	China	Longxi	Natural Phenomena	Climate, Rainfall, Drought
European Seasonal Temperature Reconstruction 1500-2004	UC-Merc	1500	2004	Europe			Natural Phenomena	Temperature, Climate
Eastern China Snow Anomaly Events and Arctic Oscillation Reconstruction	UC-Merc	5	1895	Asia	China		Natural Phenomena	Climate, Rainfall, Snow
Opium Imports and Re-Exports from Foreign Countries and British Posessions 1850-93	WHD	1850	1893	Global			Commodities	Opium, Import, Export
Import and Export of Opium Among International Commission Members 1903-1907	WHD	1903	1907	Global			Commodities	Opium, Import, Export
Opium Consumption Data for Netherlands East Indies, 1925-1938	WHD	1925	1938	Asia	Indonesia		Commodities	Opium, Consumption, Revenue, Sales
Opium Production Data for Netherlands East Indies, 1902-1938	WHD	1902	1938	Asia	Indonesia		Commodities	Opium, Production, Revenue, Sales

Table 3 (continued)

File / Data Set Name	Distr.	start date	end date	Spatial continental	Spatial national	spatial.3	Topic	variables
Silver Series from Mint and Other Records	WHD	1621	1821	North America, South America	Chile, Guatemala, Mexico, Granada, Peru, Potosi		Commodities	Silver, Mint,
Annual Silver Registrations by Cajas: Mexico	WHD	1559	1821	North America	Mexico		Commodities	Silver, Registration, Annual
Annual Silver Registrations by Cajas: Peru	WHD	1559	1821	South America	Peru		Commodities	Silver, Registration, Annual
Decennial Gold & Silver Registrations	WHD	1492	1810	North America, South America	Mexico, Granada, Ecuador, Peru, Chile, Brazil, Carribbean		Commodities	Gold, Silver, Registration
Indices of trade partner concentration for 183 countries, 1980-2008	JWSR	1980	2008	Global			Commodities	Trade, Indices, Partners, Concentration
Trade Statistics for Dahomey, 1863-1960	WHD	1863	1960	Africa	Benin		Commodities	Trade, imports, exports
Wage Data for Netherlands East Indies, 1908-1924	WHD	1908	1924	Asia			Demographic	Wages, industry, race, skill level
Diseases in Java, 1930-1937	WHD	1930	1937	Asia	Indonesia	Java	Death and Disease	Typhoid, Paratyphoid, Smallpox, Bacillary, Diphtheria, Plague, Meningitis

Table 3 (continued)

File / Data Set Name	**Distr.**	**start date**	**end date**	**Spatial continental**	**Spatial national**	**spatial.3**	**Topic**	**variables**
World Development Indicators, 1975-2000 (portion)	WHD	1975	2000	Global	Canada, France, Ghana, Malaysia, Paraguay		Other	Development Indicators
Human Development Index 1975-2000 (portion)	WHD	1975	2000	Global			Other	Development Index Trends
Tax Data (local) for Netherlands East Indies, 1910-1921	WHD	1910	1921	Asia			Demographic	Tax, Income Distribution, Local Earners
Tax Data (Dutch) for Netherlands East Indies, 1915-1929	WHD	1915	1929	Asia			Demographic	Tax, Income Distribution, European Earners
Railway Data for Netherlands East Indies, 1921-1929	WHD	1921	1929	Asia			Commodities	Rail, Freight
Wage Data For Netherlands West Indies (Suriname), 1915-1920	WHD	1915	1920	Asia			Demographic	Wages, Plantations, Absenteesim
British Opium Imports and Re-exports, 1890-1898	WHD	1890	1898	Europe	Great Britain		Commodities	Opium, Imports, Re-exports
British Opium Imports and Re-exports, 1880-1889	WHD	1880	1889	Europe	Great Britain		Commodities	Opium, Imports, Re-exports
British Opium Imports and Re-exports, 1869-1879	WHD	1869	1879	Europe	Great Britain		Commodities	Opium, Imports, Re-exports
Opium Imports and Re-Exports to/from British Possessions 1850-1893	WHD	1850	1893	Global			Commodities	Opium, Imports, Re-exports, Possessions
Rice Production and Price Data for Java, Netherlands East Indies, 1856 & 1857	WHD	1856	1857	Asia	Indonesia	Java	Commodities	Rice, Price, Production

4.3.1 Lessons Learned

From the collaboration with CHIA, we have learned what functionality in Col*Fusion is missing or need to be improved/changed.

In particular, we found that very often the DO values of many relationship's links were much lower than expected which meant that number of matching values in related columns in linked datasets were small even though the columns were describing the same entity. Related to the same issue, merged datasets had few tuples. The reason for above mentioned problem to exist is the equi-join that we used to match values and its strict requirement to the values to match exactly. For example, given two values "C.S. Lewis" and "C. S. Lewis" (notice the first string does not have a space character after first dot character) that come from two related datasets, the equi-join will be unable to identify that two values represent the same person and thus might be matched. In general this problem is known as record linkage or entity reconciliation [45][17][42][63][75]. Various methods for such problem have been proposed, however most of the time the solution is time and computationally expensive, based on heuristics and might require user involvement. Depending on the task at hand (e.g. exploratory what-if analysis), approximate solutions might be more efficient and appropriate.

The second lesson that we have learned is that users do not only want to search for specific variables and see whole datasets as the result, but they also want to search for a specific value and see the records that are only related to that value. For example, limit the search results for the records related to a specific location (e.g. continent, county, or street name, etc.) or date and time, etc.

In the next chapter I describe the problem related to the first lesson in more detail and provide solutions and algorithms for various domain types of the attributes which are used for the join. Particularly, I focus on approximate string, spatial and temporal joins.

The solution to the second lesson is left to the future work. It does not require much research effort and can be solved by indexing actual data values and extending schema graph to store them.

5.0 FUSING DATASETS

Combining data from two or more datasets is a complex procedure that usually requires high expertise and involves sophisticated algorithms to deal with structural (e.g. incompatible schemas) and semantic (data sources have different ways of referring to the same real-world entity) heterogeneities. Numerous schema matching and schema mapping algorithms have been proposed in the literature to resolve the structural heterogeneity. They range from graphical user interfaces (e.g., [80][11]), and high-level declarative languages (e.g., [26][146]) to specify schema mappings, and more recently, the use of data examples (e.g., [6][148]) to design and understand schema mappings. In previous sections, I also showed how to resolve the structural heterogeneity by developing methods to seamlessly store heterogeneous data without any limitation and loss (Section 3.1), perform efficient virtual integration (Section 3.2) and exploration (Section 3.3) thereof. However, the topic of semantic heterogeneity was not covered in details yet.

To resolve semantic heterogeneity, previous works mainly focus on detecting when records from different sources refer to the same real-world entity. This problem, known as reference reconciliation, has received significant attention in the literature, and its variations have been referred to as record linkage [187], merge/purge [87], de-duplication [157], reference matching [127], and object identification [176]. Despite the large number of works and long lasting interest in both academic and industrial research on data integration, little effort has been put towards merging data based on spatial and temporal dimension.

In this chapter I focus on the problem of merging datasets ad-hoc during query execution, i.e., join operations for which indexing or secondary access paths are not available or appropriate. Particularly, I focus on various types of approximate join techniques and their applicability limits depending on the type of data of the join attributes. While it might be impossible to find exact correspondence of two records (i.e., to do the equi-join), depending on the task at hand, the *approximate join* would suffice. Thus, the main goal of the approximate join is to provide the best effort to match corresponding tuples.

The chapter is organized as follows. Section 5.1 covers preliminaries. In Section 5.2 I first describe related similarity join techniques that are applicable when string and tree-like structures are joined. Then in Section 5.3 I describe how the similarity join approach can be used to approximately join spatial data based on the subsumption hierarchy. Finally, in Section 5.4 I describe the problem and provide novel methods to fuse temporal datasets.

5.1 PRELIMINARIES

In this section, I define the terminology that is used later on. As described in Section 3.1 heterogeneous datasets submitted by users are stored without changes in a relational store (MySQL). Thus, when operating on the datasets we can utilize terminology and operations from relational database area.

A $relation\ schema$ is represented as $R = (A_1, \dots, A_n)$, where A_i is an attribute with a domain Ω_i. A $tuple$ over schema R is a finite sequence of values that for every A_i contains a value $v_i \in \Omega_i$. An instance of the schema R is called a $relation$, r, and is a finite set of tuples over R. $r[A_i]$ or $r.A_i$ denote attribute A_i of the relation r. Consecutively, $r[A] = r.A = \{A_1, \dots, A_n\}$. I sometimes use the term table for relation, row or record for tuple, and column or variable for attribute. In what follows, I refer to two relations r and s defined over schemas $R = (A_1, \dots, A_n)$ and $S = (B_1, \dots, B_m)$ respectively.

Figure 28 shows two sample relations $Employee$ and $Manages$.

Employee

EmpName	Dept
Ron	Ship
George	Ship
Ron	Mail

Manages

Dept	MgrName
Load	Ed
Ship	Jim

Figure 28: Two sample relations $Employee$ and $Manages$

While the relational algebra defines a number of operators to manipulate data modeled as a relation, for the task of fusing datasets I focus only on the join operator. The goal of the join is to combine tuples form two relations. Below I provide definitions of most common types of join operator: Cartesian product, inner equi- and theta-join, left-, right- and full-outer join.

Definition 1 – Cartesian Product: The Cartesian product (also known as cross join), $r \times s$, of two relations r and s is defined as follows.

$$r \times s = \{z^{n+m} | \exists x \in r\ \exists y \in s\ ($$
$$z[A] = x[A] \wedge z[B] = y[B])\}$$

The second line of the definition sets the attribute values of the result tuple z to the concatenation of the attribute values of x and y.

□

The schema of z is $Z = (A_1, \dots, A_n, B_1, \dots, B_m)$ or as a shorthand $Z = (A, B)$.

In other words, Cartesian product of two relations will combine each tuple from the r relation with each tuple from the s relation by concatenating their attributes. Figure 29 shows the result of Cartesian product of the relations $Employee$ and $Manages$.

$Employee \times Manages$

EmpName	**Dept**	**Dept**	**MgrName**
Ron	Ship	Load	Ed
Ron	Ship	Ship	Jim
George	Ship	Load	Ed
George	Ship	Ship	Jim
Ron	Mail	Load	Ed
Ron	Mail	Ship	Jim

Figure 29: Result of the query $Employee \times Manages$

Definition 2 – Theta Join: The theta join ($\theta - join$) of two relations r and s on attributes $A' \subseteq A$ and $B' \subseteq B$ and predicate θ is defined as follows.

$$r \bowtie_{r.A'\theta s.B'} s = \{z^{n+m} | \exists x \in r \; \exists y \in s \; ($$
$$z[A] = x[A] \wedge z[B] = y[B] \; \wedge$$
$$x[A'] \, \theta \, y[B'])\}$$

The second line of the definition sets the attribute values of the result tuple z to the concatenation of the attribute values of x and y. The third line ensures that only those tuples are concatenated that satisfy the predicate θ.

□

The definition of the theta join can also be expressed via the selection (σ) operation as $r \bowtie_\theta s = \sigma_\theta(r \times s)$.

In other words, the result of the theta join operation consists of all the combination of tuples from r and s relation that satisfy θ. When the operator θ is the equality operator (=) then this join is also called an equi-join.

Figure 30 shows the result of the equi-join of relations $Employee$ and $Manages$ based on the equality predicate of $Employee.Dept$ and $Manages.Dept$ attributes. When the names of the attributes for the theta predicate are the same in both relation, the equi-join can be written as $Employee \bowtie_{Dept} Manages$.

$Employee \bowtie_{Dept} Manages$

EmpName	**Dept**	**Dept**	**MgrName**
Ron	Ship	Ship	Jim
George	Ship	Ship	Jim

Figure 30: Result of the query $Employee \bowtie_{Dept} Manages$

The result of the inner join (theta or equi-join) consists only of matching tuples: the tuples that satisfy the θ predicate. An outer join contains the result of inner join plus not matched tuples of one relations (or both) extended with the "fill" values for each of the attributes of the other relation. The “fill” denotes the value that is not known, which in practice corresponds to the NULL in SQL. I will use ω symbol to denote the “fill” value.

Three outer join operators are defined as left outer join, right outer join, and full outer join. The left outer join preserves all tuples from the relation on the left of the join operator. The right outer join is symmetric to the left outer join, the full outer join is defined as the union of the left and the right outer joins (the union operator eliminates the duplicate tuples). Below only left outer join definition is provided.

Definition 3 – Left Outer Join: The left outer join of two relations r and s on attributes $A' \subseteq A$ and $B' \subseteq B$ and a predicate θ is defined as follows.

$$
\begin{aligned}
r ⟕_{r.A'\theta s.B'} s = \{z^{n+m} \mid \exists x \in r\ \exists y \in s\ (& \\
& z[A] = x[A] \wedge z[B] = y[B]\ \wedge \\
& x[A']\ \theta\ y[B'])\ \vee \\
& \exists x \in r\ \forall y \in s\ (\neg(x[A']\ \theta\ y[B']) \Rightarrow z[A] = x[A] \wedge z[B] = \ \omega)\}
\end{aligned}
$$

The first three lines of the definition correspond to the theta join on matching tuples. The last line handles the case where no matching tuple y is found and thus sets the attribute values of the result tuple z to the concatenation of the attribute values of x and the null value ω.

□

Figure 30 shows the result of the left outer joins of relations $Employee$ and $Manages$ based on the equality predicate on $Employee.Dept$ and $Manages.Dept$ attributes, $Employee ⟕_{Dept} Manages$.

$Employee ⟕_{Dept} Manages$

EmpName	**Dept**	**Dept**	**MgrName**
Ron	Ship	Ship	Jim
George	Ship	Ship	Jim
Ron	Mail	ω	ω

Figure 31: Result of the query $Employee ⟕_{Dept} Manages$

In what follows I describe various approximate join techniques depending on the data type of the attributes that are used for the theta predicate.

5.2 STRING APPROXIMATE JOIN

Historically, approximate join techniques have been mostly focused on similarity based joins for string values.

5.2.1 Problem

Very often the values that do not match may be simply spelled in slightly different ways or have typos. Such situations can be quickly discovered and resolved by relaxing strict requirement of equi-join for values to have exact match. Consider two tables A and B presented on Figure 32 (the tables are taken from [10]). The numbers next to each name are the lengths of the strings. As mentioned above, in Section 5.1, when we perform a regular (equi-) join in database management system (DBMS), the tuples from the two joining tables match only if the values of attributes on which join is based match exactly. For example, if we join A and B based on the $name$ attribute, $A \bowtie_{Name} B$, equi join will result in a table with only two tuples shown in the Figure 33.

A

ID	**Name**
1023	Frodo Baggins$_{13}$
21	J. R. R. Tolkien$_{16}$
239	C.S. Lewis$_{10}$
863	Bilbo Baggins$_{13}$

B

ID	**Name**
948483	John R. R. Tolkien$_{18}$
153494	C. S. Lewis$_{11}$
494392	Frodo Baggins$_{13}$
799294	Bilbo Baggins$_{13}$

Figure 32: Example tables A and B

A.ID	**A.Name**	**B.ID**	**B.Name**
1023	Frodo Baggins	494392	Frodo Baggins
863	Bilbo Baggins	799294	Bilbo Baggins

Figure 33: Result of equi-join $A \bowtie_{Name} B$

The two other tuples are not present in the result because the string “J. R. R. Tolkein” does not exactly (character by character) match the string “John R. R. Tolkein” even though they might refer to the same person, but spelled differently. As long as at least one character is not the same, two strings are considered to be different, that is why “C.S. Lewis” does not match with “C. S. Lewis” (notice the first string does not have a space character after first dot character).

5.2.2 Problem Solution: Approximate String Equality

In many cases it is possible to state with some level of confidence that two strings represent the same entity. Similarity join is a join that performs matching based not on the exact equality of the values of the matching attributing, but based on similarity or difference measure of comparing values.

For example, *string edit distance* (sed) can be used to join the two tables from the example above. One of the most commonly used string edit distance was introduced by Levenshtein [114] and is defined as the minimum number of edit operations that transform one string into another. For example, sed(“J. R. R. Tolkein”, “John R. R. Tolkein”) = 3 by replacing “o” with “.” and deleting “hn” in second string. sed(“C.S. Lewis”, “C. S. Lewis”) = 1 by inserting space “ “ character in the first string after first “.”.

Thus, the string similarity join can be represented as a theta join where theta predicate operates on two strings and evaluates to true if and only if the similarity-based distance of the two strings is below a given threshold. For example, to perform the similarity join of tables A and B from Figure 32 on the $name$ attribute with string edit distance as defined above and a threshold equals to 3, the query can be defined as a theta join with predicate $a\,\theta\,b \equiv sed(a,b) \leq 3$. The result of such query, $A \bowtie_{sed(A.Name,\ B.Name) \leq 3} B$, is presented in Figure 34.

$$A \bowtie_{sed(A.name,\ B.name) \leq 3} B$$

A.ID	**A.Name**	**B.ID**	**B.Name**
1023	Frodo Baggins	494392	Frodo Baggins
21	J. R. R. Tolkien	948483	John R. R. Tolkien
239	C.S. Lewis	153494	C. S. Lewis
863	Bilbo Baggins	799294	Bilbo Baggins

Figure 34: Similarity join result with string edit distance and threshold 3

Another popular approach to perform string similarity join is to use token based techniques (e.g., tokens are based on either individual words or phrases for long strings, or q-gram for short strings) and set similarity metrics (e.g. Overlap similarity, Jaccard Similarity, or Dice Similarity). [10] provides a good overview of the similarity join approaches and their possible implementations in relational database management systems. Since computing similarity between two strings adds additional execution cost, many works focus on techniques and algorithms for efficient similarity join execution on large datasets. The most popular approaches use 1) blocking/filtering techniques (e.g. [98][184]) to filter out not matching tuples before computing similarity metric; 2) distributed join computation a cluster of machines (e.g. [3][108]). A recent paper [98] provides comprehensive experimental evaluation of many string similarity join algorithms.

In my work, I use threshold based string edit distance to compute similarity scores between string values during the join execution. Since the threshold largely depends on the data and the task at hand, I let user to interact with the join result by means of a threshold slider. In [166] authors report using similarity join to find schema matching rules and duplicate values by learning similarity threshold from training data.

5.3 SPATIAL (NAME-BASED) APPROXIMATE JOIN

Previous works on spatial join queries has been mostly focused on the spatial data that is represented in geographic coordinates and regions. A number of index structures such as the R-tree [78], R+-tree [161], R∗ -tree [15], Quad-tree [156], or seeded tree [122] has been developed and utilized for efficient query answering. While some algorithms use preexisting indices, others build the them on the fly.

String values can very often represent some geographic location however no geographic coordinates might be available (especially in historical datasets the actual geographic coordinates might be missing). Thus, existing spatial join techniques cannot be easily applied. String approximate join methods cannot be applied either. Some recent works report on what’s called spatial approximate string (SAS) query (e.g., [118]), however they don’t use location names and hierarchies as the join attribute. Instead they combine spatial indices with text indices to perform search for a location based on the string similarity in location’s tags.

In this section I describe how hierarchical representation of the location attribute and tree-based similarity methods can be utilized to perform ad-hoc join queries.

5.3.1 Problem

Consider two relations, A and B, shown in Figure 35 and a query that aims to join these two relations based on the match between City and Area columns. This query could be executed in several ways. Using equi-join on both attributes would result in an empty table because we cannot match exactly any value in the City column with any value in the Area column. Even if we try to perform an approximate string join based on edit distance, the result will still be an empty table.

A

City	Precipitation
Pittsburgh	700
Philadelphia	800

B

Area	Pop.	Temp.
Allegheny County	306,211	60
Pennsylvania	1,548,000	70

Figure 35: Example of datasets for spatial join

5.3.2 Problem Solution: Named Subsumption Hierarchy Approach

The location values could be represented in a hierarchical form. For example, a simplified hierarchy to illustrate the approach could be as USA subsumes Pennsylvania territorially, it subsumes Philadelphia and Allegheny County and Allegheny County subsumes Pittsburgh. Figure 36 shows modified relations A and B.

A

Area	Precipitation
USA |-Pennsylvania |- Allegheny County |- Pittsburgh	700
USA |-Pennsylvania |- Philadelphia	800

B

City	Pop.	Temp.
USA |-Pennsylvania |- Allegheny County	306,211	60
USA |-Pennsylvania	1,548,000	70

Figure 36: Modified tables A and B with hierarchical representation of the values in the location attributes

The $tree\ edit\ distance$ (ted), that is defined as the minimum cost of a sequence of edit operations (delete, insert, rename) that transform one tree into another, can be used to calculate the similarity between location hierarchies and then based on a similarity threshold, tuples from the two relations can be joined. Let us assume that all operations have the same cost equal to 1. As an example, given a distance threshold of 1, consider joining the first tuple from A table with tuples from B table. The tree edit distance between Pittsburgh and Allegheny County is 1 since either Pittsburgh node needs to be deleted in the A table or added to the B table. Thus two tuples can be joined. However, since the difference between Pittsburgh and Philadelphia is 2, the tuples cannot be joined.

Figure 37 shows the result of the spatial approximate join between the A and B tables based on the tree edit distance with threshold 1.

City	Precipitation	Area	Pop.	Temp.
Pittsburgh	700	Allegheny County	306,211	60
Philadelphia	800	Pennsylvania	1,548,000	70

Figure 37: Result of the query $A \bowtie_{ted(City,Area)\leq 1} B$

Efficient algorithms that compute tree edit distance are known, e.g. [143].

5.4 TEMPORAL APPROXIMATE JOIN

Time is an attribute of all real-world phenomena, and thus very often the datasets than need to be merged include a temporal dimension either explicitly or implicitly. However, most of the prior techniques and systems for data integration are largely agnostic to time, and hence, they cannot be immediately applied to fuse temporal datasets.

Zhu et. al [198] discuss three types of temporal heterogeneity that need to be resolved when integrating data over time. To address the record linkage problem, several temporal models (e.g., [41][119][117]) have been proposed. To resolve temporal conflicts and consistently integrate temporal datasets, time-aware and preference-aware union operators have been proposed in [12][154]. However, all previous work has been focused mostly on record linkage problem where time dimension is used primarily to track the object evolution over time (e.g., to reconstruct employment history of a specific person).

Another area where a lot of effort to incorporate temporal domain have been ongoing for several decade is the area of temporal databases ([97][96]). Each tuple in a temporal database relation is annotated with time attributes, which are treated in a special way when a query is executed. Temporal counterparts of the relational algebra operators have been developed. The temporal Cartesian product and join operators (see [71] for review) are of the most related to us since they allow to merge two temporal datasets into one. Temporal joins are arguably the most important relational operators. This is so because efficient join processing is essential for the overall efficiency of a query processor. Various algorithms (e.g., [38][52][53][71][101][102][103][64][145][151][174][196][199]) that use specialized temporal indices and partition over a number of machines were introduced to efficiently execute temporal join in temporal databases.

While both data integration and temporal database areas consider temporal dimension to represent time when some fact is valid (e.g., a time when a person works in a company), they didn't consider the case when a relation contains *aggregated* values over some time intervals (e.g., average temperature over a course of a week), which is a common case in historical datasets. Thus, to the best of my knowledge the problem of fusing *aggregated temporal datasets* was not address in the context of approximate join.

In what follows, I first define the problem formally in Section 5.4.2, show the use cases that I focus on in Section 5.4.3 and provide description of proposed methods to solve the problem in Sections 5.4.4-5.4.7.

5.4.1 Preliminaries for Temporal Join

In this section I provide definitions and notations of terms that will be used further in the text to discuss the problem of temporal approximate join and the solutions that I developed. I reuse where possible and extend where needed terminology defined in the temporal database area.

I assume discrete $time\ domain$, Ω^T, where elements are linearly ordered. I mostly refer to an element of the time domain as a $time\ unit$ but also may use terms $time\ point$, $time\ instant$, or $chronon$ [62] interchangeably. Depending on the granularity level, examples of time unit include seconds, days, months, years, etc.

A $time\ interval$ is a contiguous set of time units and is represented by two time units $[From, To]$ which denote its inclusive $From$ (start) and To (end) time points on the underlying time axis respectively. As a shorthand, a time interval will be represented as T, and $T.From$ is the $From$ time unit of the interval and $T.To$ is the To time unit. A time unit t belongs to the time interval T, $t \in T$, iff $From \leq t \leq To$. Chronological relations between two time intervals can be expressed using the Allen's interval algebra [8], which defines a set of Boolean predicates, composition table and converse operation. Here I define two operators on two intervals that will be useful for further discussion.

$$intersect(T_1, T_2) = \begin{cases} true & if\ \exists t\ (t \in T_1 \wedge t \in T_2) \\ false & otherwise \end{cases} \tag{6}$$

$$overlap(T_1, T_2) =$$

$$\begin{cases} [\max(T_1.From, T_2.From), \min(T_1.To, T_2.To)] & if\ intersect(T_1, T_2) \\ 0 & otherwise \end{cases} \tag{7}$$

The $intersect(T_1, T_2)$ operator is true when two intervals has at least one time unit in common and $overlap(T_1, T_2)$ returns the number of time units that are in common if two intervals intersect and 0 if they don't.

Relations in temporal databases are annotated with time dimension that usually record a $valid\ time$, $transaction\ time$, or both (known as $bitemporal$ model). Valid time describes when a tuple is true in the real world, and transaction time captures when a tuple was created or altered. In what follows, I focus on relations that only record valid time. Thus, definition of temporal relation schema extends the definition of the non-temporal relation given in Section 5.1 as follows. A $temporal\ relation\ schema$ is represented as $R^T = (A_1, \dots, A_n,\ From,\ To)$, where A_i is the non-temporal (also called explicit) attribute with domain Ω_i, and $From$ and To are temporal attributes with domain Ω^T. To distinguish relations with valid-time semantics from other types that will be introduced later in the text, instances of valid-time relations are denoted as r^{VT} and the schema as R^{VT}.

Consider as example two temporal relations (taken from [71]) in Figure 38 that are representative of valid time relations. The tuples in both relations are annotated with time intervals T that denote when each tuple is valid. For example, the tuple (Ron, Ship, [1, 5]) in the Employee relation represents the fact that Ron worked for the Ship department from time 1 to time 5.

$Employee^{VT}$

EmpName	Dept	T
Ron	Ship	[1, 5]
George	Ship	[5, 9]
Ron	Mail	[6, 10]

$Manages^{VT}$

Dept	MgrName	T
Load	Ed	[3, 8]
Ship	Jim	[7, 15]

Figure 38: Two sample temporal relations with interval temporal attributes

To distinguish between a tuple of a non-temporal relation and a tuple of a temporal relation, I refer to the latter one as a $report$. The $lifespan$ or $length$ of a report $r_i \in r^T$, $|r_i|$, is the number of time units in its time interval and is calculated as:

$$|r_i| = |r_i.T| = r_i.From - r_i.To + 1 \tag{8}$$

Since time intervals lack the total ordering, the next two definitions find the report that covers the smallest (the earliest) or the largest (the latest) time unit respectively.

Definition 4 – First: Given a set of reports (or a temporal relation) r^T, the $first(r^T)$ operator is defined as follows.

$first(r^T) = r_f \iff r_f \in r^T \wedge \forall x \in r^T(r_f.From \leq x.From)$

The condition ensures that the first report is the one whose $From$ time unit is the smallest (earliest).

□

Definition 5 – Last: Given a set of reports (or a temporal relation) r^T, the $last(r^T)$ operator is defined as follows.

$last(r^T) = r_l \iff r_l \in r^T \wedge \forall x \in r^T(r_l.To \geq x.To)$

The condition ensures that the last report is the one whose To time unit is the largest (latest).

□

The $lifespan$ of a relation r^T, $|r^T|$, is the number of time units between the $From$ time unit of the earliest report and the To time unit of the latest report and is calculated as:

$$|r^T| = first(r^T).From - last(r^T).To + 1 \tag{9}$$

Note that the lifespan of a relation can contain some time units that are not covered by any reports. For example, if relation r^{VT} has two reports $r_1 = (v_1, 1,\ 2)$ and $r_2 = (v_2, 5, 8)$, where

$v_1$and v_2 are some explicit values, that cover time units [1, 2] and [5, 6, 7, 8], the lifespan would be 8 time units, however time units [3, 4] are not covered. Such intervals of time units are called $gaps$. The Definition 6 defined the $gaps$ operator that returns true if there are time units that are not covered by any report in the given set of reports.

Definition 6 – Gaps: Given a set of reports (or a temporal relation) r^T, the $gaps(r^T)$ operator is defined as follows.

$$gaps(r^T) = \begin{cases} true & \exists t \in [first(r^T).From, last(r^T).To] \forall x \in r^T (t \notin x.T) \\ false & otherwise \end{cases}$$

The condition to return true checks whether there exists a time unit in the lifespan of the set of the reports that doesn't no belong to all reports' time intervals

□

As mentioned earlier, temporal relational algebra operations have been developed to operate on temporal relations. Two most relevant operations to the discussion in this chapter are the temporal Cartesian product ($\times^T$) and the temporal Theta-join ($\bowtie_\theta^T$). The two definitions below are taken from [71] and are important because they allow to understand how valid-time semantics treat the value of explicit attributes.

Definition 7 – Temporal Cartesian Product [71]: The temporal Cartesian product, $r^{VT} \times^T s^{VT}$, of two valid time temporal relations r^{VT} and s^{VT}, defined over schemas $R^{VT} = (A_1, \dots, A_n,\ From,\ To)$ and $S^{VT} = (B_1, \dots, B_n,\ From,\ To)$ respectively, is defined as follows.

$$r^{VT} \times^T s^{VT} = \{z^{(n+m+2)} \mid \exists x \in r\ \exists y \in s\ ($$

$$z[A] = x[A] \wedge z[B] = y[B] \wedge$$

$$z[T] = overlap(x[T], y[T]) \wedge z[T] \neq 0)\}$$

The second line of the definition sets the explicit attribute values of the result tuple z to the concatenation of the explicit attribute values of x and y. The third line computes the timestamp of z and ensures that it is nonempty. □

Definition 8 – Temporal Theta Join [71]: The temporal theta join, $r^{VT} \bowtie_{\theta}^{T} s^{VT}$, of two valid time temporal relations r^{VT} and s^{VT}, defined as in Definition 7, is defined as follows.

$$r^{VT} \bowtie_{\theta}^{T} s^{VT} = \sigma_{\theta}(r^{VT} \times^{T} s^{VT})$$ □

For example, the result of the temporal Cartesian product ($\times^{T}$) of the Employee and Manages relations from the Figure 38 is shown in Figure 39. The query $Employee^{VT} \times^{T} Manages^{VT}$ can be expressed in English as "Show the names of employees and managers where the employee worked for the company while the manager managed some department in the company" [71].

$Employee^{VT} \times^{T} Manages^{VT}$

EmpName	Dept	Dept	MgrName	T
Ron	Ship	Load	Ed	[3, 5]
George	Ship	Load	Ed	[5, 8]
George	Ship	Ship	Jim	[7, 9]
Ron	Mail	Load	Ed	[6, 8]
Ron	Mail	Ship	Jim	[7, 10]

Figure 39: The result of the query $Employee^{VT} \times^{T} Manages^{VT}$

Note that the non-temporal Cartesian product would match every tuple from the Employee relation with every tuple of the Manages relation and the result would have six tuples. The result of the temporal Cartesian product, however, basically represents the join of the two relations based on the intersections of the time intervals T. In fact, in the original work by Segev and Gunadhi

[159][160] on the temporal Cartesian product, the operator was named $time\ join$, and the abbreviation $T - join$ was used.

An example of a temporal equi-join of the $Employee^{VT}$and $Manages^{VT}$ relations based on the $Dept$ attributed, $Employee^{VT} \bowtie^{T}_{Dept} Manages^{VT}$ is shown in Figure 40.

$$Employee^{VT} \bowtie^{T}_{Dept} Manages^{VT}$$

EmpName	**Dept**	**Dept**	**MgrName**	**T**
George	Ship	Ship	Jim	[7, 9]

Figure 40: The result of the query $Employee^{VT} \bowtie^{T}_{Dept} Manages^{VT}$

All temporal operators operate on explicit (non-temporal attributes) while taking into account temporal dimension to produce only valid tuples. In the next section, I will describe when the existing approaches to temporal data join do not solve the problem that we encounter while building the advance historical data integration infrastructure.

5.4.2 Problem

The temporal join operators presented in the previous section (Section 5.4.1) work on temporal data successfully because of the semantic of the $valid\ time$ data model. The $valid\ time$ temporal relation contains facts that are $valid$ (and are the same) at each time unit of the time interval. For example, the $Employee^{VT}$ relation from Figure 38 can be represented as a non-temporal relation (see Figure 41 for the excerpt of the converted relation) where the TU attribute stands for time unit and can be treated as a regular non-temporal attribute. Non-temporal relational algebra operators can be used to operate on such table. In fact, temporal operators act as though they are non-temporal operators applied independently at each time unit.

$Employee$

EmpName	Dept	TU
Ron	Ship	1
Ron	Ship	2
Ron	Ship	3
Ron	Ship	4
Ron	Ship	5
George	Ship	5
George	Ship	6
…	…	…

Figure 41: Excerpt of the valid time temporal relation $Employee^{VT}$ converted into non-temporal relation $Employee$

Very often, however, we need to deal with another type of temporal data, which was not considered previously in the temporal database area. Instead of reporting constant facts that are valid during some time intervals, this type of relations reports aggregated value of some real life phenomena over some period of time (time interval). I call such relations as $Aggregate\ Time$ relations.

Definition 9 – Aggregate Time Relation: Given a non-temporal relation r defined over schema $R = (C_1, \dots, C_n,\ V_1, \dots, V_m,\ TU)$ where each tuple describes some object/entity by a set of categorical attributes $C_1, \dots, C_n$ that do not change over time and a set of variable attributes $V_1, \dots, V_m$ values for each are recorded at each time unit TU. Given a set of time intervals TI where each interval is a defined over subset of time units from the lifespan of TU such that the values of

$C_1, \ldots, C_n$ attributes are same for each time unit. And given an aggregate function $f: \mathbb{R}^k \to \mathbb{R}$, the aggregate temporal relation, r^{AT}, is defined as follows.

$$r^{AT} = \{z^{(n+m+2)} \big| \, \forall t \in TI \, \exists r' \subseteq r \, (\forall x \in r' \, (x.TU \in t) \, \wedge$$
$$(\exists y \in r' \, (z[C] = y[C])) \, \wedge$$
$$\left(\forall i \in 1 \ldots m \left(z.V_i^f = f(r'.V_i)\right)\right) \wedge$$
$$z.T = t)\}$$

In the first line, for every time interval t a temporary relation r' is defined that contains a subset of tuples of r whose TU belongs to the interval t. The second line sets the values of the constant categorical attributes $C_1, \ldots, C_n$ of the result tuple z as a copy of any tuple of the temporary relation r'. The third line applies the aggregate function f to each variable attribute V_i of r' and sets the result into corresponding attribute of the result tuple z. The last line sets the time interval of the result tuple z as the time interval t.

□

As it follows from the definition, the schema of the aggregate time relation is $R^{AT} = (C_1, \ldots, C_n, V_1^f, \ldots, V_m^f, \ From, \ To)$.

As an illustration of the Definition 9 consider the example shown in Figure 42. The $Temperature$ relation, called original or ground truth relation, describes two stations St1 and St2 (we call them entities or objects) and the temperature values measured at each station at each time unit. The $Temperature^{AT}$ relation is the aggregate time relation derived from $Temperature$ by applying mean function over time intervals TI within each entity.

$$TI = \{[1,2],[3,4]\} \qquad f \equiv mean$$

$Temperature$

Station	**Temp**	**TU**
St1	10	1
St1	15	2
St1	20	3
St1	15	4
St2	7	1
St2	10	2
St2	8	3
St2	5	4

$Temperature^{AT}$

Station	**Temp$^{\text{Mean}}$**	**T**
St1	12.5	[1, 2]
St1	17.5	[3, 4]
St2	8.5	[1, 2]
St2	6.5	[3, 4]

Figure 42: Illustration of non-temporal relation Temperature and its aggregate time version $Temperature^{AT}$

The first notable difference between aggregate time and valid time relations is that in the case of aggregate relation, the values of the variable attributes (e.g. Temp$^{\text{Mean}}$ attribute) at each time unit are unknown and thus we cannot split the time interval into time units and assign each time unit the same value. For example, slicing the first report in the $Temperature^{AT}$ relation into two tuples with the same Temp$^{\text{Mean}}$ value will produce the tuples (St1, 12.5, 1) and (St1, 12.5, 2) which do not correctly describe the true values of the Temp attribute in the Temperature relation.

To simplify further discussions without the loss of generality, I focus on aggregate time relation that describes only one object and one variable attribute of that object. Thus, a shortened schema, $S = (From,\ To,\ V)$, will be used. We can do so, because joining on categorical attributes is straightforward operation and standard join approaches can be used. The temporal fusion, that will be discussed later, is only needed for relating variable attributes within each object. The superscripts will be omitted if it is clear from the context what type of relations is used. I call variable attribute as a variable.

Consider two relations $AvgTemperature$ and $AvgCloudiness$ (Figure 43) that hold average temperature and average cloudiness values respectively from some time unit $Form$ to time unit To. Notice that the time intervals intersect both within one relation and between two relations. The time intervals can also be represented graphically as shown in Figure 44.

$AvgTemperature$

From	To	Temp
1	4	15
3	5	10
6	10	20

$AvgCloudiness$

From	To	Cloud
1	5	4
7	10	8

Figure 43: Sample data from two datasets describing two variables – average temperature and average cloudiness. Observations are temporally overlapping both within and between datasets

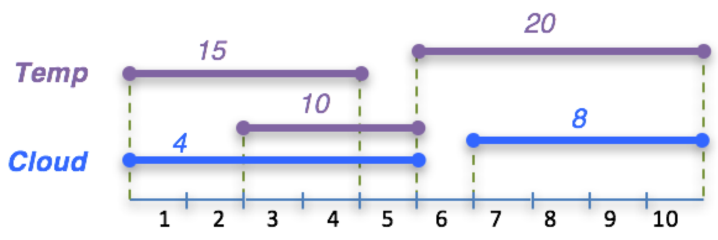


Figure 44: Interval representation of time interval overlaps for $AvgTemperature$ and $AvgCloudiness$ tables from Figure 43

Consider a research question that would require average temperature and average cloudiness to be combined into one table: what temperature corresponds to what cloudiness value? In database terms, such operation can be performed with a join query.

Using traditional equi-join on the $Form$ and To attributes of the $AvgTemperature$ and $AvgCloudiness$ tables from Figure 43 would yield an empty table since there are no matching time intervals in the two tables that match exactly on their ends. Other join predicate operators would find matching tuples, but they will not recover the complete picture and it is not always clear which combination of them to use. Moreover, the result still might be empty if there are gaps in the data. As it was shown earlier, it is also not possible to use temporal join approaches from the temporal database area since they require valid time semantics.

5.4.3 Taxonomy of Aggregate Time Relations and Reports Characteristics

As mentioned above, datasets that are developed independently come in different forms and shapes. In this section I describe five main characteristics of aggregate time relations and reports (see Table 4 for the summary):

- $Coverage$ – characteristic of a relation that describes whether the relation has gaps as defined in Section 5.4.1 - time intervals that are not covered by any report in the relation.
- $Intersection\ within\ relation$ – characteristic of a relation that describes whether the relation has reports that intersect or not.
- $Time\ interval\ length$ – characteristic of a relation that describes whether the reports in the relation are short (aggregation is performed over few time units) or long (aggregation is performed over large number of time units).

- $Varying\ inteval\ length$ – characteristic of a relation that describes whether all reports in the relation have the same length or not.
- $Aggregation\ type$ – the type of the aggregation function: average value (also called index series), sum of the values (also called flow series, e.g., exports, production, household consumption) or a particular point in time (e.g., the value of the first or last time unit in the report time interval) (also called stock series, e.g., unemployment, money stock, public sector debt).

$Coverage$, $Time\ interval\ length$ and $Varying\ inteval\ length$ characteristics are also applicable to the valid-time relation type.

Combination of values of different characteristics is called a $scenario$. For example, Figure 44 shows an instance of the scenario when reports have partial coverage (see gaps for time unit 5 for $Temp$ variable and for 6 for $Cloud$ variable), intersection within relation (two reports in the $AvgTemperature$ relation intersect), and varying time interval length. Figure 45 shows an example of a scenario with complete coverage, no intersection within relation, varying interval length and mixed short and long reports. Figure 46 shows example of a scenario with complete coverage, no intersection within relation, and constant interval length short reports.

Table 4: Temporal categorization criteria

$Coverage$	Full; Partial
$Intersection\ within\ relation$	Yes; No
$Time\ interval\ length$	Small; Large
$Varying\ inteval\ length$	Yes (varying); No (constant)
$Aggregation\ type$	Average (index); Sum (flow); First or Last observation in time interval (stock)

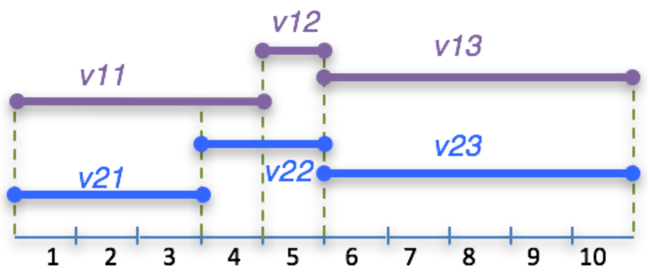


Figure 45: Example of a scenario with complete coverage, no intersection within relation, varying interval length and mixed short and long reports

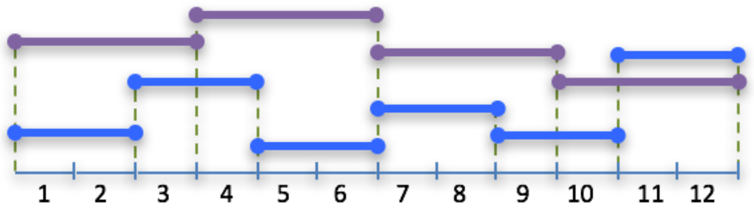


Figure 46: Example of a scenario with complete coverage, no intersection within relation, constant interval length short reports

Note that even though Figure 46 shows ideal scenario within each relation, non-temporal or temporal valid time equi-join approaches are still not applicable since there are no exact matches on reported time intervals between two relations. In what follows I describe methods to perform join for such scenarios.

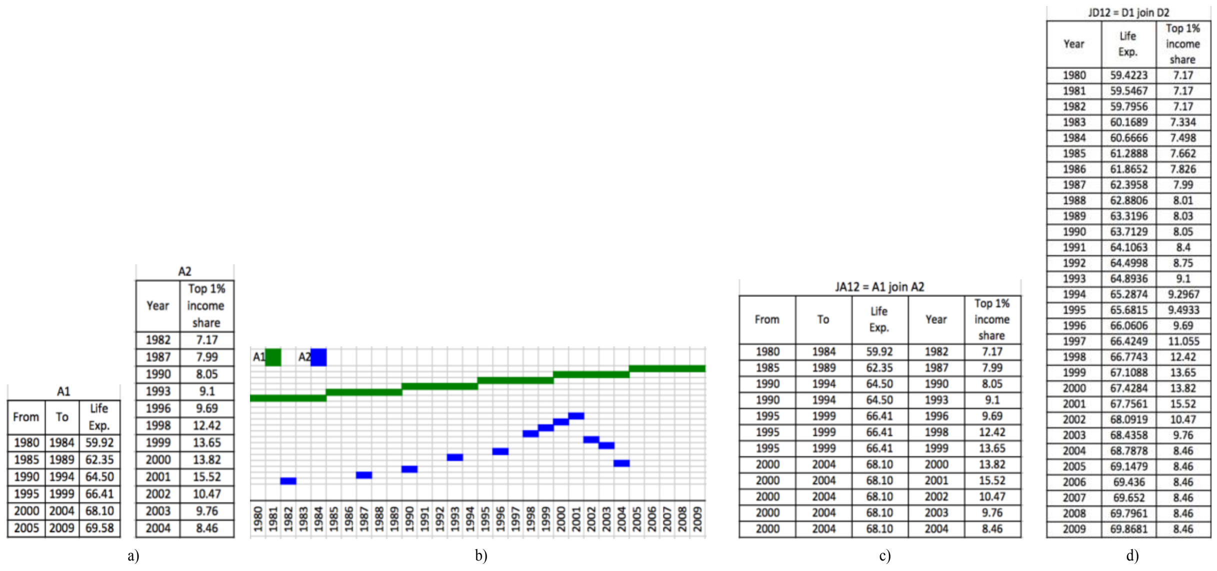
5.4.4 Overview of Join Strategies of Aggregate Time Relations

As I mentioned earlier, performing the standard equi-join on temporal datasets, such as in Figure 44, would yield an empty table when time units don't match exactly. While it is impossible to find exact correspondence for two aggregate time variables on each time unit in the same way how it is done in case of valid time relation, depending on the task at hand, an *approximate join* would suffice. Thus, the goal of approximate join of two aggregate time tables is to provide the best effort to match corresponding tuples.

In general, we can apply two high level strategies to join two aggregate time relations A_1 and A_2 that were derived from the original relations by applying some aggregation function:

1. Reports in each relation can be disaggregated to a common time scale and then the standard non-temporal equi-join can be applied. We call such approach $Disaggregate\ Join$ (explained in more detail in Section 5.4.5).
2. Alternatively, join can be performed directly on the reports based on the temporal distance between those reports. We call such approach $Aggregate\ Join$ (explained in more detail in Section 5.4.6).

Both approaches are illustrated in Figure 47. The example shows (a) aggregate relation A1 with average life expectancy statistics reported every five years and aggregate relation A2 with annual Top 1% income share statistic reported at different times; (b) graphical representation of the aggregate relations A1 (green) and A2 (blue); (c) join result JA12 of the relations A1 and A2 based on Aggregated Join method; (d) disaggregated relations D1 and D2 as well as Disaggregated Join result based on the exact match of time units (years). The process of merging aggregated data streams is resource consuming and it involves trade-offs between accuracy of the produced results, execution time, and consumed computational resources.



A1

From	To	Life Exp.
1980	1984	59.92
1985	1989	62.35
1990	1994	64.50
1995	1999	66.41
2000	2004	68.10
2005	2009	69.58

A2

Year	Top 1% income share
1982	7.17
1987	7.99
1990	8.05
1993	9.1
1996	9.69
1998	12.42
1999	13.65
2000	13.82
2001	15.52
2002	10.47
2003	9.76
2004	8.46

a)

A1
A2
1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

b)

JA12 = A1 join A2

From	To	Life Exp.	Year	Top 1% income share
1980	1984	59.92	1982	7.17
1985	1989	62.35	1987	7.99
1990	1994	64.50	1990	8.05
1990	1994	64.50	1993	9.1
1995	1999	66.41	1996	9.69
1995	1999	66.41	1998	12.42
1995	1999	66.41	1999	13.65
2000	2004	68.10	2000	13.82
2000	2004	68.10	2001	15.52
2000	2004	68.10	2002	10.47
2000	2004	68.10	2003	9.76
2000	2004	68.10	2004	8.46

c)

JD12 = D1 join D2

Year	Life Exp.	Top 1% income share
1980	59.4223	7.17
1981	59.5467	7.17
1982	59.7956	7.17
1983	60.1689	7.334
1984	60.6666	7.498
1985	61.2888	7.662
1986	61.8652	7.826
1987	62.3958	7.99
1988	62.8806	8.01
1989	63.3196	8.03
1990	63.7129	8.05
1991	64.1063	8.4
1992	64.4998	8.75
1993	64.8936	9.1
1994	65.2874	9.2967
1995	65.6815	9.4933
1996	66.0606	9.69
1997	66.4249	11.055
1998	66.7743	12.42
1999	67.1088	13.65
2000	67.4284	13.82
2001	67.7561	15.52
2002	68.0919	10.47
2003	68.4358	9.76
2004	68.7878	8.46
2005	69.1479	8.46
2006	69.436	8.46
2007	69.652	8.46
2008	69.7961	8.46
2009	69.8681	8.46

d)

Figure 47: Illustration of high level strategies for temporal approximate join of two aggregate time relations

Figure 48 shows schematic representation of the problem setup and high level view of the possible approaches to join aggregate time temporal relations.

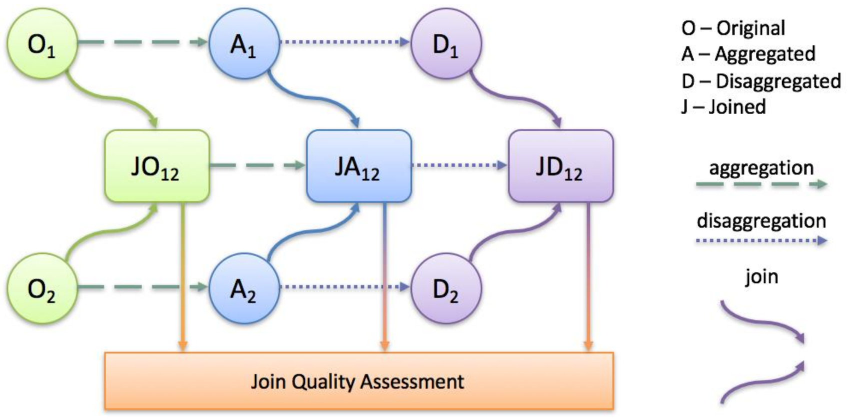


Figure 48: Schematic representation of high level view of the approaches for temporal approximate join of two aggregate time relations

5.4.5 Disaggregate Join of Aggregate Time Relations

As it was mentioned earlier, under the disaggregate join strategy we first disaggregate each aggregate time relation into a relation that contains estimated values for each time unit and then use standard equi-join.

Given a disaggregation function, $dis(r,\ Ir) \rightarrow r'$, that takes an aggregate time relation with schema $R^{AT} = (From,\ To,\ V)$ and a set of items Ir that represent external knowledge, and produces a relation r' with schema $R'^{AT} = (TU,\ V')$, we can define disaggregate join as follows.

Definition 10 – Disaggregate Join: The disaggregate join, $r^{AT} \bowtie^{TDJ} s^{AT} \mid Ir, Is$, of two aggregate time temporal relations r^{AT} and s^{AT} given two sets of external knowledge elements $\{Ir_1, \ldots Ir_k\}$ for r^{AT} and $\{Is_1, \ldots Is_l\}$ for s^{AT} is defined as follows.

$r^{AT} \bowtie^{TDJ} s^{AT} \mid Ir, Is = dis(r, Ir) \bowtie_{TU} dis(s, Is)$ □

The external knowledge set can contain anything that a disaggregate function can use to do better estimation. For example, it can be distribution of the variable, some context information, etc. The quality of the disaggregate join solely depends on how accurate a disaggregation function can estimate values for each time. Below I provide three disaggregation methods that can be used as a disaggregation function in the disaggregate join approach.

5.4.5.1 Temporal Disaggregation The Temporal Disaggregation approach is based on the temporal disaggregation methods that are used in the time series analysis of mostly economic data (see [39], [29] and [158] for review). Briefly, given a low frequency time series (e.g. annual sales, weekly stock market index, etc.) the goal of temporal disaggregation is to produce a high-frequency series (e.g. quarterly sales, daily stock market index, etc.) while satisfying temporal aggregation (additivity) constraint. Temporal aggregation constraint ensures that either the sum, the average, the first or the last value of the resulting high frequency series is consistent with the low frequency series. If available, related series observed at the required high frequency can be used to disaggregate the original observations. Such series are called *indicators*. However, care must be taken when selecting indicators since two strongly correlated low frequency time series may not be strongly correlated at a higher frequency [36]. Thus, choosing good indicator series is not a very straightforward task.

Temporal disaggregation methods have been studied extensively in the area of econometrics and statistics and many methods to perform the disaggregation have been proposed. Smoothing, model based and statistical methods have been developed that can be organized into three groups as follows (note, the list of methods is not exhaustive):

- Univariate Without Indicators: If no higher frequency indicator is available then a smoothing or model based methods can be used, such as: low-pass interpolation (interpolation followed by low-pass smoothing by means of a filter (e.g. Hodrick-Prescott filter [152]); Boot-Feibes-Lisman (BFL) smoothing method [27]; Stram-Wei (an ARIMA model-based method) [167].
- Univariate With Indicators: When a higher frequency indicator is available a range of statistical methods can be used, such as: Denton [51]; Denton-Cholette [44]; Chow-Lin [74]; Fernández [66]; Litterman [121]; Santos Silva-Cardoso (dynamic extension of Chow-Lin) [163]; based on MIDAS regression [76].
- Multivariate With/Without Indicators: Multivariate models are used to model and explain the interactions and comovements among a group of time series variables: Multivariate Denton; Rossi [153]; Di Fonzo [68]; Polynomial method [195]; SUTSE [133].

Methods without indicator series solely rely on the data points in the low-frequency series while estimating missing values for higher frequencies; methods with indicators are primarily concerned with movement preservation, generating a series that is similar to the indicator. While univariate methods derive one high-frequency series from one low-frequency one, the multivariate methods derive multiple high-frequency series from multiple low-frequency series. Thus, in the multivariate case, the estimated high-frequency series must fulfill both temporal and contemporaneous aggregation constraints.

Some of the most popular temporal disaggregation methods have been implemented in R by Sax and Steiner [158], Ecotrim by Barcellan et al. [14] and as Matlab extension by Quilis [177].

Figure 49 shows an example of applying BFL, Fernandez and Chow-Lin temporal disaggregation methods to disaggregate annual series into quarterly series. The top left plot visualizes true quarterly sales that were recorded for each year from 1975 to 2010. The true sales series is used as the ground truth to evaluate the performance of the three disaggregation methods. The top middle plot visualizes the annual sales series (annual sale for one year is the sum of sales of 4 quarters for that year). The top right plot visualizes quarterly exports series that will be used as an indicator series since the sales and exports have similar behavior. The annual series and the indicator series are what is available as input to the disaggregation method.

The bottom left plot shows the result of the BFL method that doesn’t use the indicator series to perform disaggregation of the annual sales series into quarterly one. The bottom middle and right plots visualizes the results of Fernandez and Chow-Lin methods respectively. Both methods used the quarterly export series as an indicator series. All three bottom plots also report root mean square error (RMSE) between the true quarterly sales series and the estimated quarterly series produced by each method.

As we can see from the Figure 49, the BFL method simply smoothens the annual series while the other two methods rely on the dynamics of the indicator series and are able to recover smaller fluctuations of the quarterly sales series that were not present in the annual series. Hereafter, I am only going to use BLF and Fernandez method as two representative methods of univariate temporal disaggregation without and with indicator series respectively.

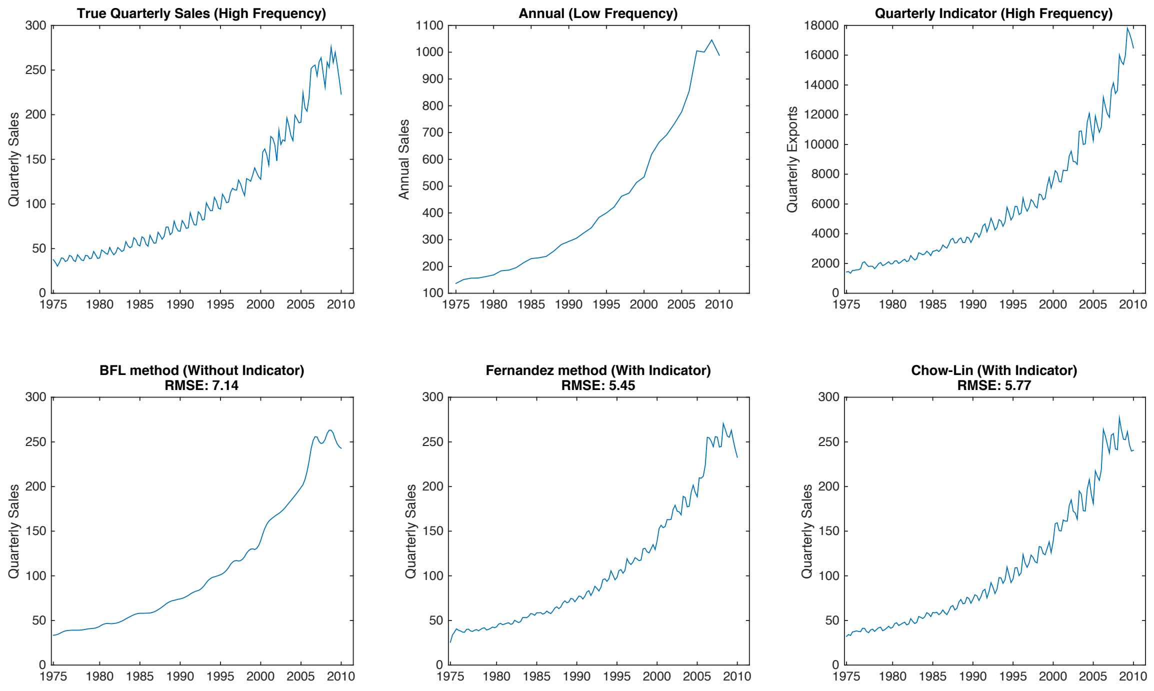


Figure 49: Example of applying BFL, Fernandez and Chow-Lin temporal disaggregation methods to disaggregate annual series into quarterly

5.4.5.2 Polynomial or Spline Interpolation Since the task of the disaggregate function is to estimate values for unknown time units based on the values of the known time units, standard mathematical interpolation methods, such as polynomial or spline interpolation methods, can be used.

While this method is conceptually simple, it doesn't satisfy the temporal additivity constraint in the case when mean function is used to produce aggregated reports. It also cannot be directly applied when there are intersections of reports within one relation. Finally, this method is not applicable when sum aggregation function is used to produce aggregated reports.

5.4.5.3 Spread, Aggregate, Fill, Extend (SAFE) Heuristic I propose this method as a heuristic that distributes the value of the report to each time unit in case of the index or stock aggregation function or the value divided by the number of time units in the case of the flow aggregation function (similar to the 1/n disaggregation method). Whenever there are intersecting reports it calculates average value to resolve the conflict and propagate that value to the other time units. Note that this method doesn't not satisfy additivity constraint.

Figure 50 shows an example of the SAFE method applied to slightly modified $Temp$ variable from Figure 44.

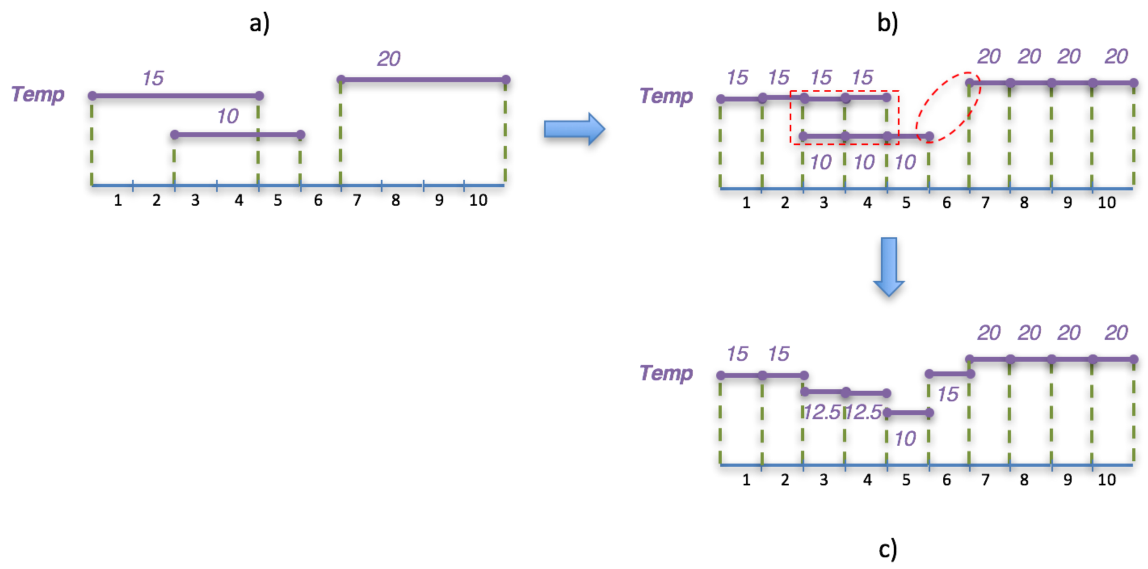


Figure 50: Example of applying the SAFE method

Figure 50(a) shows three reports that present average value of the Temp variable. Two reports intersect, and there is a gap between second and the third report. Therefore, TD or Polynomial disaggregation methods cannot be applied directly. Figure 50(b) shows that the value on each time unit was estimated by dividing the reported values equally (the *spread* phase of the method). Red dashed rectangle and oval shows the conflicting and gap time units. The *aggregate* phase resolves the conflicting time units by computing average values and the *fill* phase resolves the gap time unit by computing an average value of the adjacent not empty time units. The *extend* phase is not shown in the figure but would propagate the values outside of the given time units is needed. Figure 50, (c) shows the final outcome of the SAFE method.

Each of the three method is best applicable in different scenarios and have various limitations going from the Temporal Disaggregation to Polynomial to SAFE method. The univariate temporal disaggregation methods as the disaggregation functions for the temporal approximate join have three main applicability limitations. First, they cannot be directly applied when relations have partial coverage (e.g., not applicable to the $AvgCloudiness$ relation in Figure 44). This limitation can be resolved by applying disaggregate function at complete sections of the relation separately, however this increases join complexity since complete sections first need to be identified. The second limitation requires all reports in a relation to have the same length (e.g., not applicable to either of the relation in Figure 45). Third, they cannot be applied when there are intersections of reports within a relation (e.g., not applicable to the $AvgTemperature$ relation in Figure 44). The multivariate methods are not applicable in general in our case since they require contemporaneous aggregation constraints and thus can be only performed on homogeneous time series that are not independent from each other, whereas in the case of a join operation the relations to be joined are completely independent from each other.

The second method, Polynomial or Spline interpolation, has fewer restrictions, namely it is not applicable when sum aggregation function was used to produce low frequency series and it is applicable only when each relation doesn't have intersecting reports. However, since the method doesn't preserve temporal additivity, the quality of the result might be lower.

The last method, SAFE, has no applicability restrictions, however the quality of the join might be very low since the method is very simplistic.

Figure 51 and Figure 52 show comparison of two temporal disaggregation (BFL and Fernandez), polynomial (8th degree) and SAFE methods to disaggregate low frequency series (annual and triennial respectively) into quarterly (high frequency) series. Thus, the only difference between input data in the two figures is the length of the aggregation. The annual series is produced by averaging the sales of 4 quarters (i.e., one year), the triennial series is produced by averaging the sales of 12 quarters (i.e., three years). Note the difference with the Figure 49 in which annual sales represented the sum of quarterly sales. I could not use the sum annual series, because the polynomial interpolation method doesn't work for the sum aggregation function.

The top left plot visualizes true quarterly sales that were recorded for each year from 1975 to 2010. The true sales series is used as the ground truth to evaluate the performance of the four disaggregation methods. The top middle plot visualizes low frequency series (annual in Figure 51 and triennial in Figure 52). The top right plot and the bottom plots from left to right visualize disaggregated series produced by BFL, Fernandez, 8th degree polynomial interpolation and SAFE methods respectively.

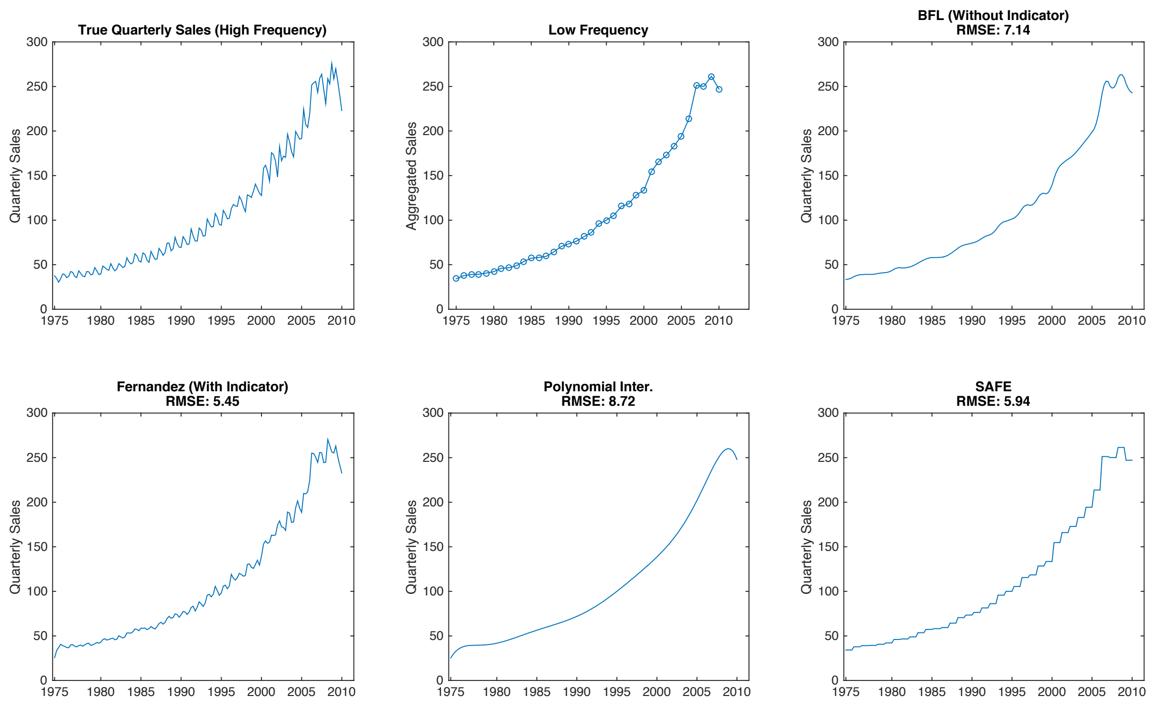


Figure 51: Comparison of two temporal disaggregation (BFL and Fernandez), polynomial (8th degree) and SAFE methods to disaggregate annual (low frequency) series into quarterly (high frequency) series

As we can see from the figures, the BFL and Polynomial methods are trying to smoothen the low frequency series by estimating unknown high frequency point between known low frequency points and thus highly resemble the low frequency series. The Fernandez method relied on the indicator series (it is not show in Figure 51 and Figure 52, but it is the same as the one shown in top right plot in Figure 49). Quality of all methods decreases in Figure 52 because the aggregation lengths is greater and thus more unknown high frequency points need to be estimated. Most vividly the decrease in quality is shown by the SAFE method since this method doesn't perform any statistical computation and doesn't rely on any information besides the value of the current report that need to be disaggregated.

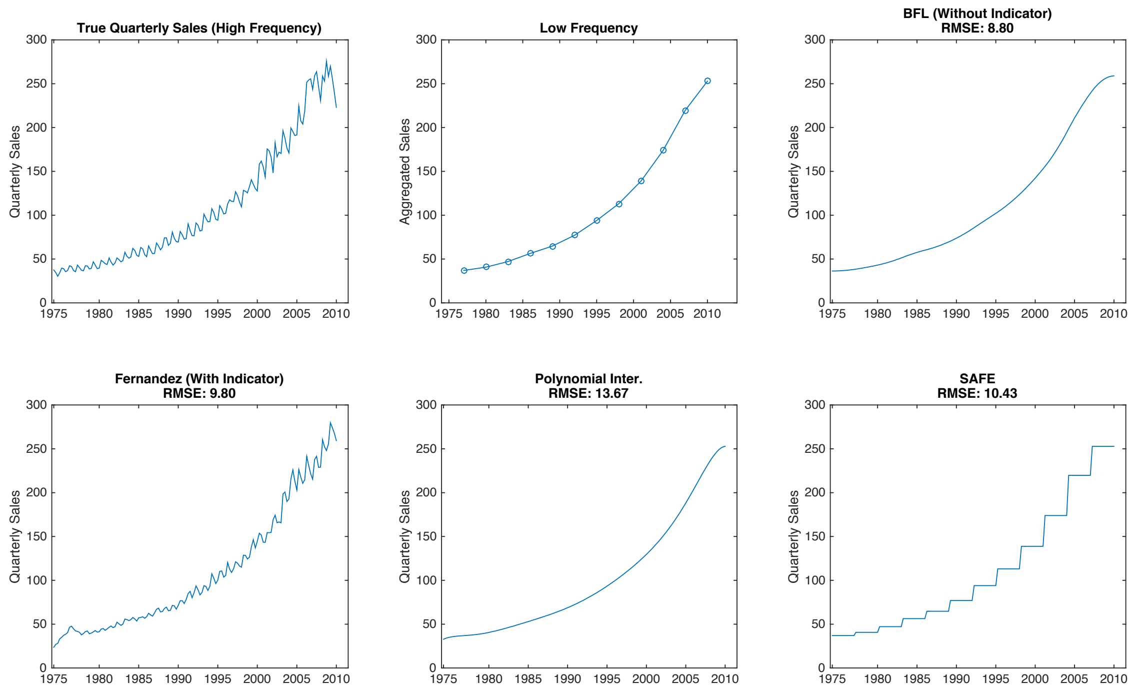


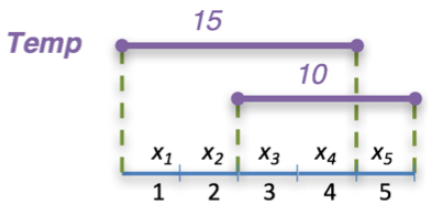
Figure 52: Comparison of two temporal disaggregation (BFL and Fernandez), polynomial (8th degree) and SAFE methods to disaggregate triennial (low frequency) series into quarterly (high frequency) series

Now it is time to put the temporal disaggregation methods into the context of the temporal approximate join that we are trying to solve. The general idea is to represent aggregated reports $A_1$and A_2 (from Figure 48) as low-frequency time series and then to disaggregate them into high-frequency series D_1 and D_2 by using most suitable disaggregation method. The result of the disaggregation would be two time series with estimated values for matching time units. Standard equi-join techniques then could be used to merge the two series to construct a joint table (JD_{12}) as explained in Definition 10.

The temporal disaggregate join approach is conceptually similar to the unfold/fold operator in IXSQL [123][49] language that first splits time intervals into time units, then applies the non-temporal join operator and finally collapses the value-equivalent tuples over the consecutive time units into time intervals. However, the fold/unfold method only works for valid-time relations.

5.4.5.4 Empirical Evaluation In this section I describe the experimental evaluation of the disaggregate join methods explained in previous sections. If not said otherwise, the experiments were run on the Mac Book with Processor 2.4 GHz Intel Core i7 and 8Gb 1600 MHz DDR3 memory.

In addition to comparing above methods with each other, I also compare them to the Reverse Substitution (RS) method developed in [112] to fuse intersecting reports within one relation. In a nutshell, the idea of the RS method is to represent intersecting reports' intervals as unknown values x and find them by solving a linear system $Ax = b$ where the rows of the observation matrix A correspond to reports covering time intervals of x and b is the vector of values of the reports. To solve the system, author applied nonnegative least square method and showed that the solution for the system can be found as $x = A^T(AA^T)^{-1}b$. Since my goal is to find values for each time unit, x represents the unknown values of the variable on each time unit. For example, for the first two intersecting reports of the $Temp$ variable in Figure 44, the underdetermined linear system in the matrix form will be as in Figure 53.



$$\begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 15 \\ 10 \end{bmatrix}$$

Figure 53: Example of applying RS method to find values on each time unit

To evaluate the quality of disaggregation methods, e.g., how accurate can a disaggregation method estimate the value on a higher frequency given the value on a low frequency, I use relative distance measure that is defined as below:

$$Relative\ Distance\ (RD) = \frac{1}{n}\sum_{i=1}^{n} \frac{|estimated\ value_i - actual\ value_i|}{\max(estimated\ value_i, actual\ value_i)} \qquad (10)$$

where n is the number of value points on high frequency.

The relative distance measure is mainly concerned with how close on average the estimated value is to the actual value on each time unit. To estimate the quality of a join, we could also utilize the relative distance and evaluate each variable separately. However, doing so doesn't not capture how the combination of the values of two variables in the join is close to the combination of values in the ground truth. Additionally, depending on the task at hand, relaxed requirement on the time component or on exact value correspondence might be acceptable. For example, given two ground truth tuples as in Figure 54 left and a join tuple as in Figure 54 right, using value based quality metrics, like relative distance, would result in a high error since the join tuple values are not very close to the ground truth tuple on the same time unit. However, if we relax the requirement on the time component, the tuple on the right is "closer" to the second tuple on the left.

Ground Truth

TU	V1	V2
1	1	2
2	3	4

Join

TU	V1	V2
1	4	4

Figure 54: Sample ground truth and join tuples to illustrate tuple scale similarity score

To find which tuples from the join result are "close" to which tuples in the ground truth, I calculate similarity scores between a join tuple and all ground truth tuples and select the highest one. Multiple similarity metrics have been developed in the literature. I evaluated two metrics, namely Manhattan Distance and Cosine Similarity, and their modifications that I will describe next. Each tuple can be considered as a three component vector. Remember, that we want the metrics to be in the range from 0 to 1, where 0 is the least similar and 1 is identical, and take into account weights for each component. Thus, the weighted Manhattan Similarity (MS) is calculated as below.

$$MS(\bar{v}_1, \bar{v}_2, \bar{w}) = 1 - \frac{abs(\bar{v}_1 - \bar{v}_2) \cdot \bar{w}}{\max\_d} \tag{11}$$

where $\bar{v}_1, \bar{v}_2$ are the vectors for which similarity score is calculated, $\bar{w}$ is the vector of weights (all three vectors must have the same number of components), $\max\_d$ is the maximum distance between all vectors in a relation, $abs(\bar{x})$ returns a vector with absolute values of each component of the vector $\bar{x}$.

Standard Cosine Similarity doesn't consider components' weights and is sensitive to the vectors magnitude, e.g., $cos\_sim([1,1,1],[1,1,2])$ is different from $cos\_sim([1,1,3],[1,1,2])$, but in our case the similarity value should be the same in those two cases since they only different by 1 in the third component. Therefore, I introduce a modified version, that we call Canonical Cosine Similarity (CCS), which is calculated as follows:

$$CCS(\bar{v}_1, \bar{v}_2, \bar{w}) = \frac{\hat{\imath} \cdot \bar{v}_{12}}{\|\hat{\imath}\| \, \|\bar{v}_{12}\|} \tag{12}$$

where $\hat{\imath}$ is the unit vector, $\bar{v}_{12} = \hat{\imath} + abs(\bar{v}_1 - \bar{v}_2) \circ \bar{w}$, $\circ$ is the Hadamard product.

While the Canonical Cosine Similarity improves on standard cosine similarity in terms of our requirements, it has one problem. The vectors that are different in magnitude but have exactly the same direction will have perfect similarity score of 1. For example, $CCS([2,2,2],[3,3,3],[1,1,1]) = 1$, which is not what we need. To fix that problem, I calculate the final version of the similarity measure, called Canonical Cosine Similarity Norm Scaled (CCSNS) that I use in my work as follows:

$$CCSNS(\bar{v}_1, \bar{v}_2, \bar{w}) = CCS(\bar{v}_1, \bar{v}_2, \bar{w}) * \left(1 - \frac{\|\hat{\imath} - \bar{v}_{12}\|_1}{\max(\|\hat{\imath}\|_1, \|\bar{v}_{12}\|_1)}\right) \tag{13}$$

where CCS is defined in (12) and $\hat{\imath}$ and $\bar{v}_{12}$ are defined as in (12).

To illustrate the behavior of the four similarity metrics, Manhattan Similarity, Cosine Similarity, Canonical Cosine Similarity and Canonical Cosine Similarity Norm Scaled, I have generated all possible permutation of values between 1 and 10 with step 2 of three components of two vectors and then calculated similarity scores between those vectors. Figure 55 shows the result with weight vector as [1, 1, 1]. On the x axis is the difference (–) between two vectors and on y axis is the corresponding similarity score. For example, for two vectors [1, 1, 1] and [1, 1, 1], the different is [0, 0, 0] and thus we can find it in the middle of the x axis and then see that the similarity score is 1 on the y axis. The next to the right x tick label is for the case when the difference between vector 1 and 2 on the second component is two, e.g., [1, 3, 1] and [1, 1, 1], or [3, 7, 5] and [3, 5, 5], etc.

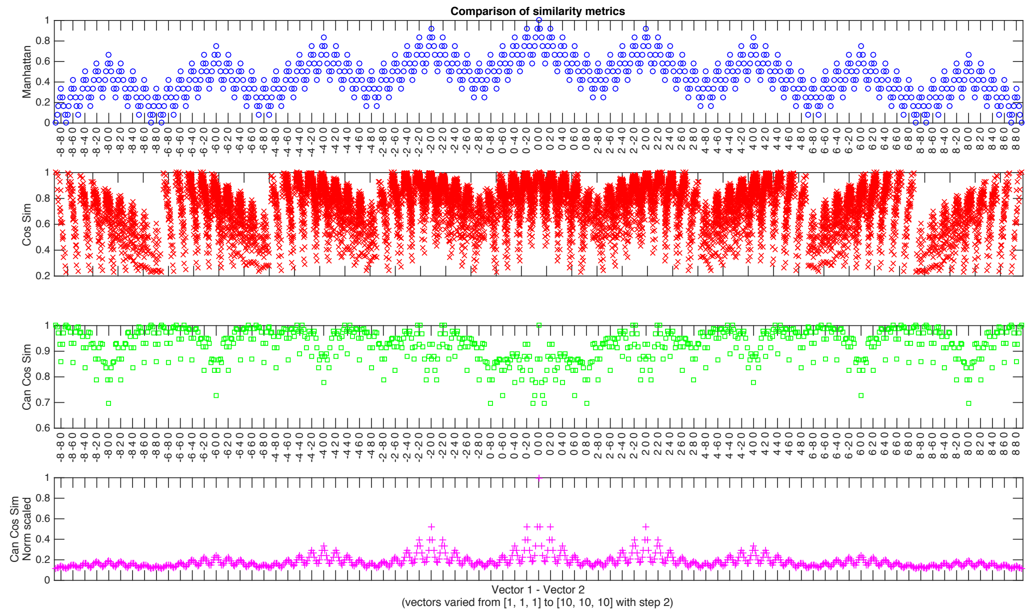


Figure 55: Comparison of four similarity metrics

The values of the weight vector change the behavior of the similarity metric. Figure 56 shows the differences in behavior for the Canonical Cosine Similarity Norm Scaled metric. The first from the top plot – all components are weighted equally; second plot – the value of second variable is not considered at all (the weight is 0); third plot – both variable are considered equally, however the time component is completely ignored; the last plot – both time and the second variable are not considered at all.

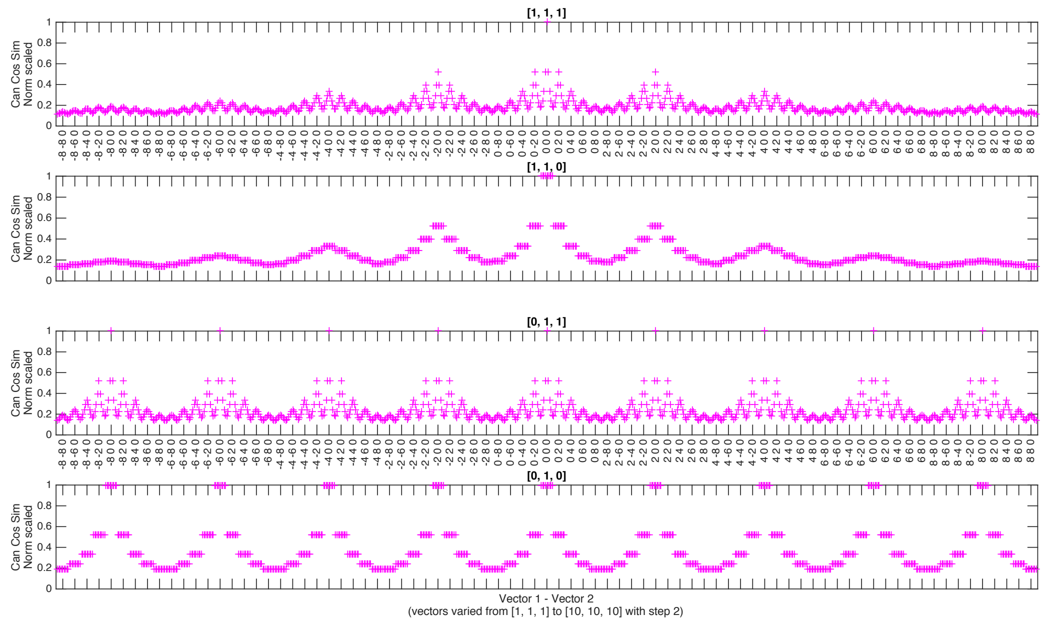


Figure 56: Behavior of the Canonical Cosine Similarity Norm Scaled metric for different weight combinations

Now, having a score for each join report, we can evaluate the whole join relation in a way similar (but with some modifications) to the precision and recall metrics in the information retrieval field. Particularly, the precision is equal to the score, and the recall is equal to the number of reports in the join whose score is greater or equal to the given score over the size of the ground truth relation. By varying a threshold of the acceptable score, the values of precision and recall will change accordingly.

5.4.5.4.1 Scalability Besides the applicability limitations described in the previous sections, I am interested to see how scalable the disaggregation methods are. I have run BFL, Fernandez, Polynomial (8th degree), SAFE and RS methods on various scenarios by varying the number of reports and report length values. Table 5 shows the combination of values for the scenarios.

Report values were drawn from the normal distribution $\sim N(100, 5)$. The lifespan of the relation (the total number of time units) was set equal to $number\ of\ reports\ * report\ length$. The reports were generated without intersections, covering the whole lifespan of the relation except for the RS method for which the lifespan of the relation was divided by two to generate some intersecting reports.

Table 5: Parameters of the scalability scenarios

Number of Reports	Report Length
10	3
10	4
10	12
10	24
10	48
10	100
100	3
100	4
100	12
100	24
100	48
100	100
1000	3
1000	4
1000	12
1000	24

Figure 57 shows the result of the experiment. Y axis shows the execution time in seconds on a log scale, x axis shows the scenarios. We can see from the figure that all methods take more time to finish when the number of reports and their length increase. However, temporal disaggregation methods, BFL and Fernandez, show considerable degradation and, for example in the case when there are 1000 reports each 24 time units long, take about 10^4 seconds (~2.78 hours) to disaggregate just one relations. I also ran the experiment on PSC computing cluster and got similar results because inherently the TD algorithms are not parallelized. Remember that we need to disaggregate two relations and then also concatenate values on corresponding time units to perform the join operation. Of course, two relations can be disaggregated in parallel, but still the performance is not acceptable for the ad-hoc join queries. Figure 58 shows the relative distance measure for the corresponding scenarios. As expected, Fernandez method showed lowest error in all scenarios because of the presence of the good indicator series. The worst performance in terms of relative distance measure was shown by the RS method due to the low number of overlapping reports. In Section 5.4.5.4.2 I explore how quality of the disaggregation methods depend on the nature of the underlying data and report scenarios.

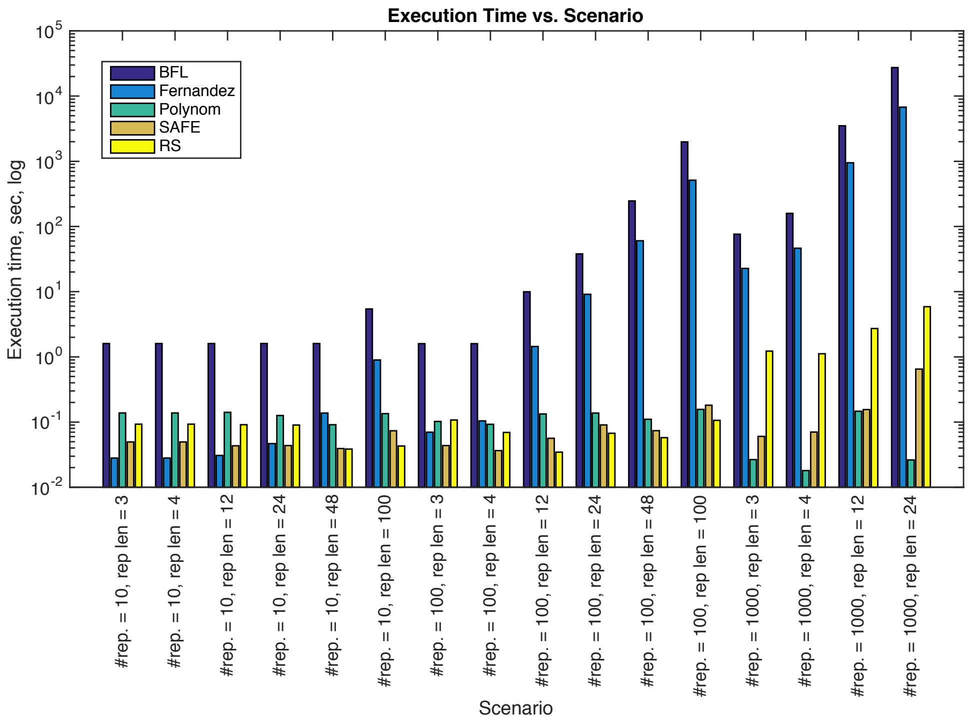


Figure 57: Execution time of each scenario

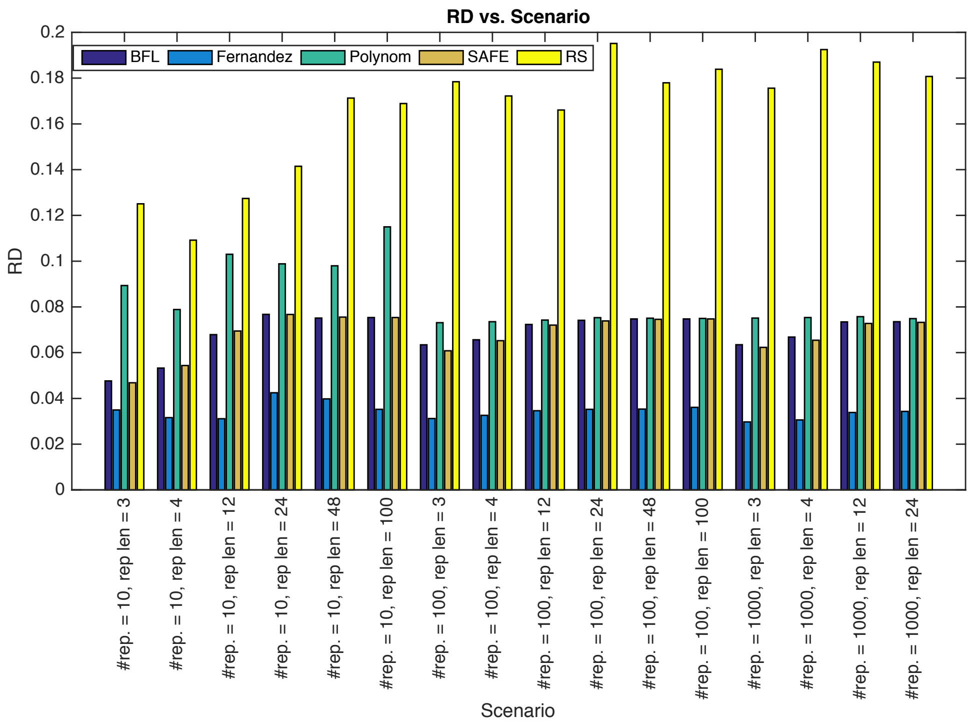


Figure 58: Relative distance quality measure versus number of parallel execution of multiple pieces of one relation

To speed up temporal disaggregation of one relation, instead of disaggregating the whole relation at once, I consider splitting the relation into shorter pieces so that each piece can be disaggregated in parallel. Figure 59 shows the result executing BLF and Fernandez temporal disaggregation methods on the scenario with 500 reports and each report of 12 time units length. The y axis shows the time it took to perform the disaggregation against the x axis that shows how many pieces were disaggregated. The time to disaggregated the whole relations was 804.87 and 124.66 seconds for BFL and Fernandez methods respectively. However, as the number of pieces into which the relation was broken and which can be executed independently (in parallel) increases, the execution time drops exponentially. For example, by breaking the relation into 10 pieces each with 50 reports reduces the execution time to under 1 second which is over two magnitude improvement compare to disaggregating the whole relation. And braking the relation into 100 pieces each with 5 reports, allows to bring the execution time down to 0.0027 and 0.0019 seconds for BFL and Fernandez methods respectively. Of course, executing 100 parallel computations might not be feasible in some cases, however even running 100 disaggregation computations sequentially lowers the execution cost, e.g., 100 * 0.0027 = 0.27 seconds, which is 2981 times improvement in case of BFL method. The improvement is due to the fact that it is much faster to deal with smaller matrices than with large ones during the disaggregation execution.

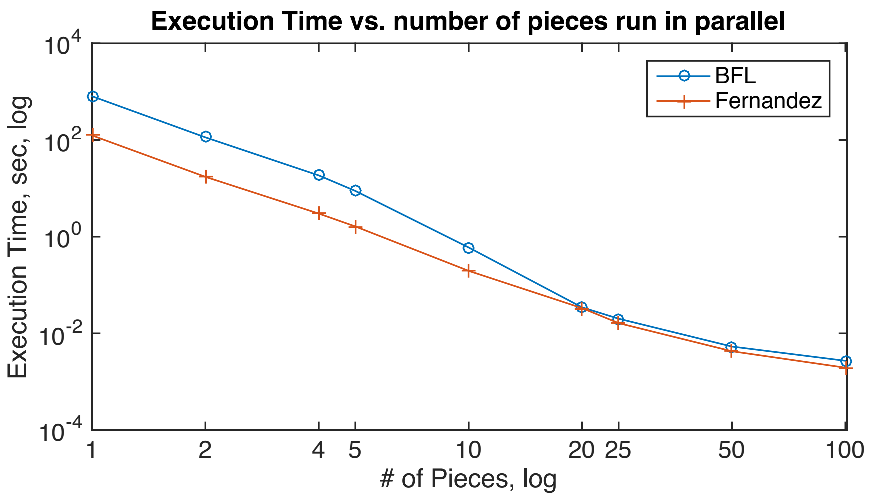


Figure 59: Execution time versus number of parallel execution of multiple pieces of one relation

The intriguing questions of course is whether breaking the relation into small pieces and performing disaggregation on each piece independently influences the quality of the disaggregation. Figure 60 shows that the quality of the BFL method didn't change a lot, while the error for the Fernandez method increased from 0.0343 for the whole relation to 0.0482 for the 100 pieces case. This is due to the the "end point" problem when forward looking observations are required, but are not available towards the end of the piece [36].

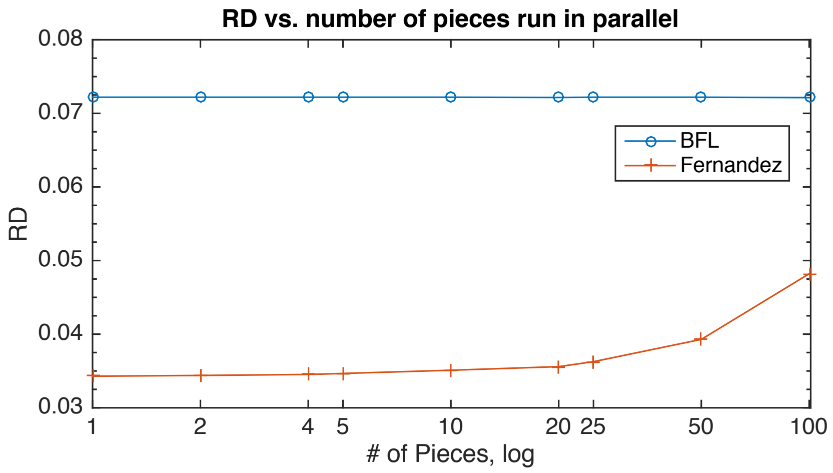


Figure 60: Relative distance quality measure versus number of parallel execution of multiple pieces of one relation

5.4.5.4.2 Disaggregation Methods Quality Comparison This simulation experiment is setup similar to the ones in [190][112] where for a given number of time units (the maximum lifespan of a relation) I vary number of measured events on each time unit (event density), number of reports, and reports' lengths. Table 6 summarizes all different scenarios that I run for lifespan of 1000 time units. All parameters are randomly generated from a normal distribution with expected value shown in the Table 6 and standard deviation equals to 5. The start time unit of each report is drawn from a uniform distribution. Each scenario is repeated 5 times and average values are reported. Figure 61 and Figure 62 show examples of generated reports for the scenario 2 and 3 respectively.

Table 6: Experiment Setup Parameter and Scenarios Description

Scenario	**Event Density**	**Number of Reports**	**Report Length**
Few short reports on sparse events	20	20	20
Few long reports on sparse events	20	20	100
Many short reports on sparse events	20	100	20
Many long reports on sparse events	20	100	100
Few short reports on dense events	100	20	20
Few long reports on dense events	100	20	100
Many short reports on dense events	100	100	20
Many long reports on dense events	100	100	100

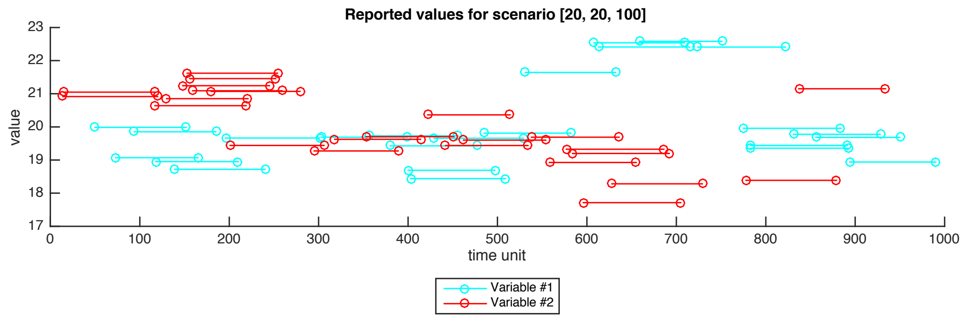


Figure 61: Example of generated reports for the scenario 2

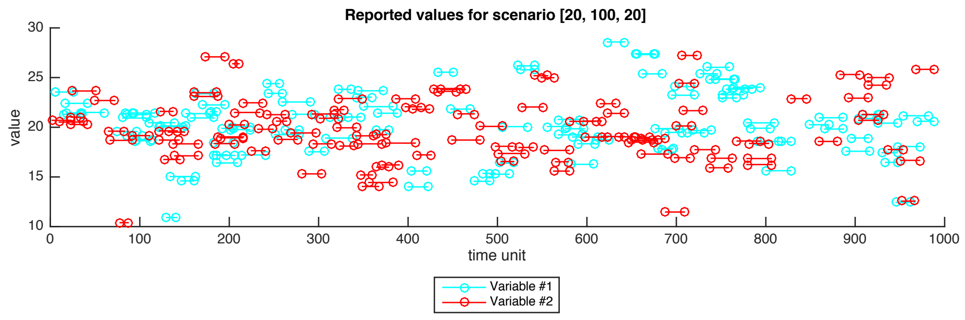


Figure 62: Example of generated reports for the scenario 3

Since depending on the scenario the generated reports might either intersect or have gaps, temporal disaggregation methods are not applicable directly. Thus I generate reports for TD methods in a slightly different way. Particularly, when new report's start and end time units intersect with any already existing report, its position is regenerated to avoid intersection. If it is impossible to generate new start and end time points so that there are no intersections (e.g. the whole lifespan of the relation is covered or the gaps are shorter then the lengths of a new report), then report generation phase is complete (even if the number of reports is less than given parameter). Then I apply TD methods on all consecutive strips of reports on each relation separately, and finally use equi join to combine reports from two relations that match on time unit value. The RS and SAFE methods are applied directly, however in this case RS method will be used to estimate values for each time unit.

Figure 63 shows the result of the experiment. Each subplot shows Precision/Recall curves for the BFL, Fernandez, SAFE and RS methods as solid lines when scores were calculated with $\overline{w} = [1,1,1]$ and as dashed lines when scores were calculated regardless of time unit component, i.e., $\overline{w} = [0,1,1]$. The ideal performance would be a horizontal line at precision 1.

There is no straightforward interpretation of the obtained results. As we can see, in general, the performance of each method depends on the the nature of the data and the scenario of how reports are obtained. All methods showed very low precision when taking time component into consideration. The best performance is shown by the SAFE and BFL methods, while Fernandez showed the worst performance. The reason for that might be the nature of the data. Since the data is generated from a normal distribution with small deviation, all data points are centered around the mean. The SAFE method performs the best because it averages the reports and thus gets closer to the ground truth mean value. Fernandez on the other hand, uses the indicator series that might add additional fluctuation which introduce higher error.

Going from the left to the right column we can see that all methods except the SAFE (which has already had a good recall), improve in recall. This is due to the fact that number and length of reports is increasing and thus more time units are covered by reports (and more reports intersect) providing more information, particularly to the RS method, to estimate values for each time unit. The SAFE method has good recall value regardless of the number of reports or report length because of the Fill and Extend phases of its computation where the estimated value of a time unit is propagated to the non-covered time units.

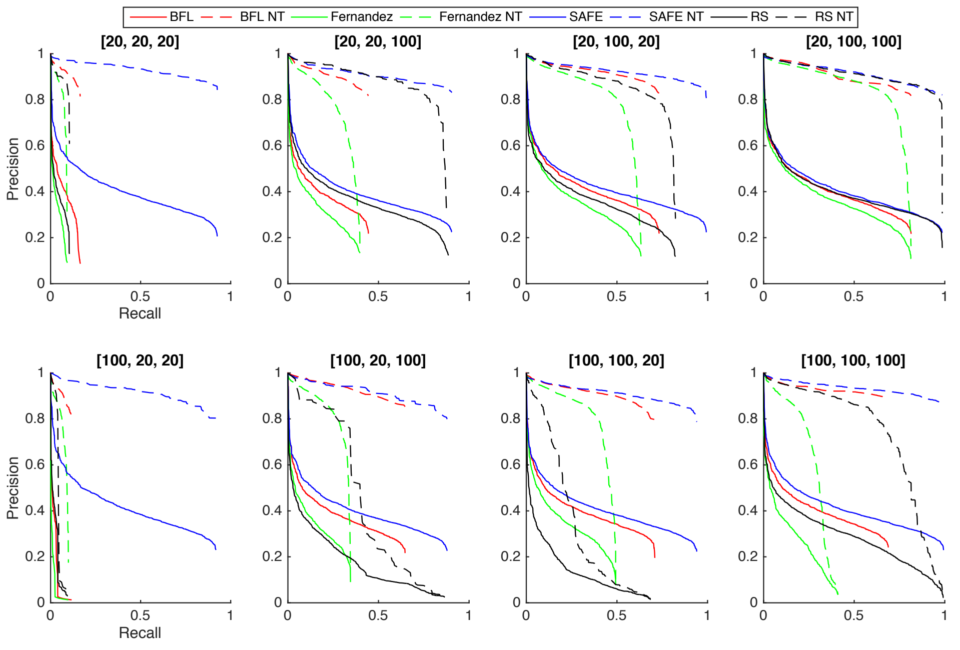


Figure 63: RP curves for sigma 5

In previous work, [112], they didn't consider other types of input data except normal distribution with deviation equal to 5. In Figure 64, I run the above experiment but with the standard deviation value set to 15 and therefore increasing the variability and spread of the ground truth values on each time unit. As the result, we can see that the performance of each method has decreased (all curves are now closer to the left bottom corner).

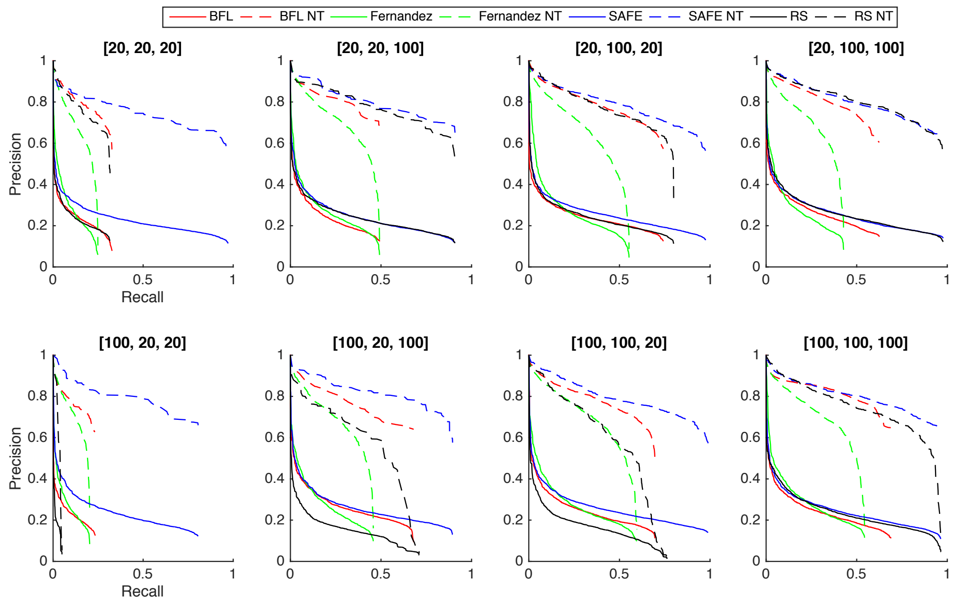


Figure 64: PR curves for sigma 15

As the last experiment, I ran the Netlogo traffic grid[2] simulation and obtained two variables changing over time by varying the available parameters of the simulation. The variables are shown in Figure 65. Figure 66 shows the result of the experiment. The performance of all methods are similar to the case with normal distribution with standard deviation set to 15 in Figure 64. I also run the experiment with a different reporting strategy when for a given report length (10, 20, 50, and 100), reports are generated without intersections covering the whole relation. Figure 67 and Figure 68 show examples of obtained reports with report length 20 and 100 time units respectively. The result is show in Figure 69.

[2] Netlogo (ccl.northwestern.edu/netlogo) is multi-agent programmable modeling environment. The traffic grid simulation is included sample model that can also be access via http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Social%20Science/Traffic%20Grid.nlogo

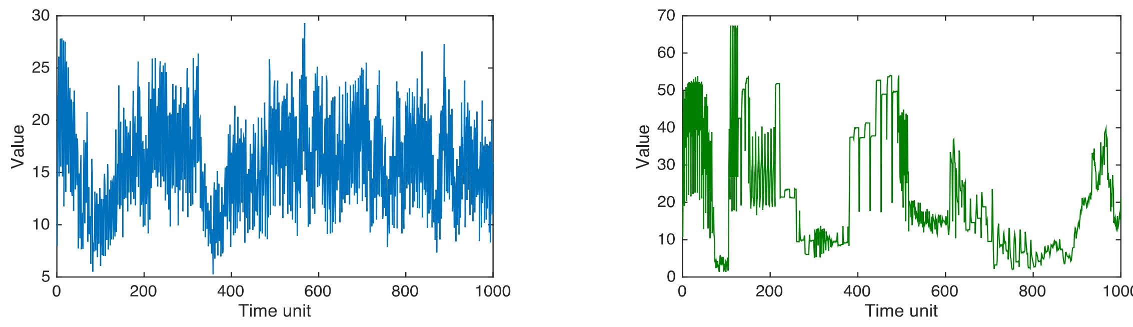


Figure 65: Two variables at the time unit scale obtained from a Netlogo simulation

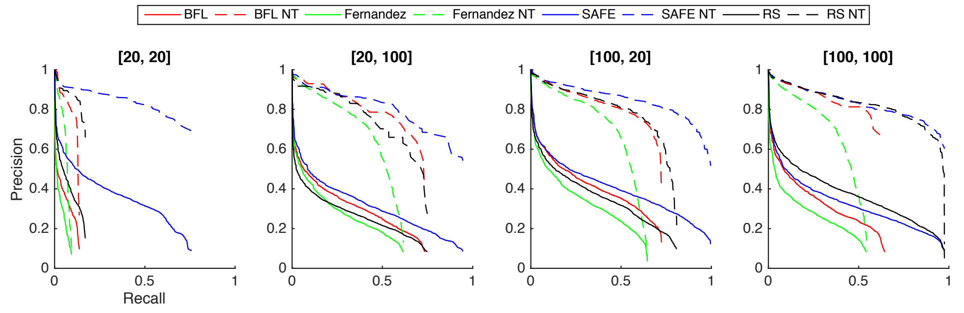


Figure 66: PR curves for the join of two variables obtained from Netlogo simulation

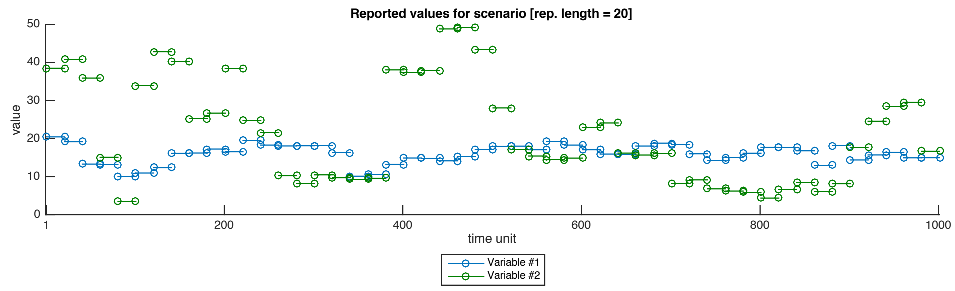


Figure 67: Example of generated reports for the variables obtained form Netlogo simulation with report length 20

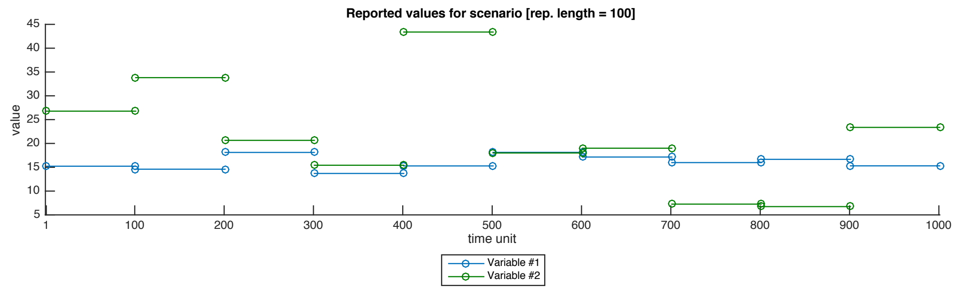


Figure 68: Example of generated reports for the variables obtained form Netlogo simulation with report length 100

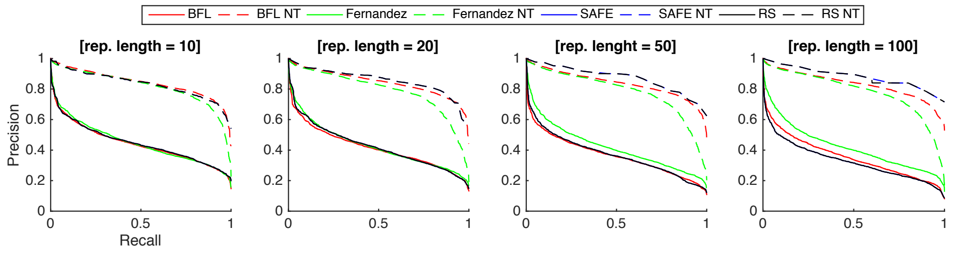


Figure 69: PR curves for the join of two variables obtained from Netlogo simulation in the case when reports cover the whole relation lifespan without intersections for various report lengths

This suggest one more time that applicability of disaggregation methods to perform join on a time unit scale is dependent on the data and also on how reports were generated. For example, Figure 70 shows result of applying disaggregation methods to a variable that have a notable value transition (the left plot). As we can see from the right plot, if there is only one report that covers all 50 time units, the difference between estimated values compared to the actual values measures in terms of RMSE is high (above 8). This is due to the fact that generating one report over the 50 time units will average the values and thus will loose the information about the transition.

However, as soon as there are two reports (or more) the error drops significantly for all methods, because now the information about the transition is preserved in the reports. SAFE and RS methods perform identically because there are no intersecting reports. Also notice that those two methods have a slight increase in error when the number of reports was 3. This is because second report covers the transition interval and its disaggregation is most inaccurate. The BFL or Fernandez don't have this problem, because BFL is based on the polynomial method and Fernandez uses indicator series that shows the transition as well. More intersecting reports help both SAFE and RS methods to produce better results.

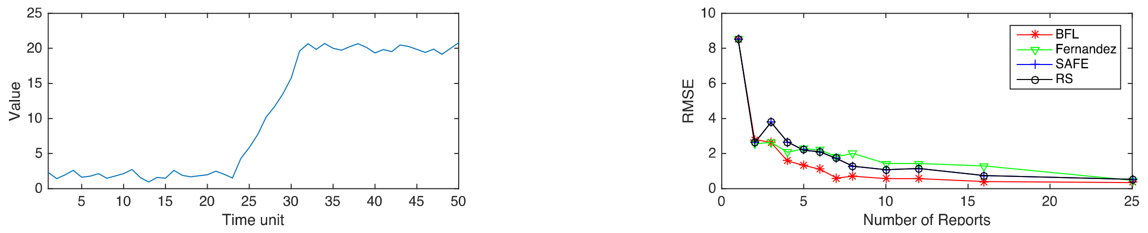


Figure 70: Example of disaggregation quality dependence on the nature of the data and the report length (number of reports)

Since in real life scenario we don't have control how the reports are generated, it is hard to suggest the best method to use or predict the quality of the join. In addition, it is not always the goal to find corresponding values of two variables on each time unit, but instead a join on aggregated reports is needed that doesn't do any estimation and preserve the exact values. In the next section, I will talk about an aggregate join approaches that do right that.

5.4.6 Aggregate Join of Valid and Aggregate Time Relations

In previous sections the goal was to perform a join of two aggregate time relations by trying to disaggregate each relation independently first and then do equi-join. Given two relations r^{AT} and s^{AT} that are defined over schemas $R^{AT} = (From, To, V_1)$ and $S^{AT} = (From, To, V_2)$ respectively, the goal was to obtain a relation j with schema $G = (TU, V_1, V_2)$. The task, as the experiments in Section 5.4.5.4 show, turned out to be highly dependent on the nature of the input data. The quality of one or the other join method depends on many parameters and there are no obvious dominating values of those parameters that would guide the join algorithm to guarantee certain level of accuracy of the join result. In addition, the join quality requirement depends on the task at hand and for some tasks lower quality but faster execution is acceptable, whereas for other task the quality is much more important. Finally, the goal to reach the $G = (TU, V_1, V_2)$ schema is not what the actual goal of the database join is.

In this section, I introduce a different strategy, an aggregate join, its goals and approaches, and how to implement it. Overall join strategy still falls into the schematic representation show in Figure 48 and the taxonomy of datasets reports is still relevant, however now I consider cases when joining valid time relation with valid time or with aggregate time relation, or joining two aggregate time relations.

As it was mentioned earlier, in the aggregate join strategy reports from two relations are joined directly based on their time intervals only. Thus, the temporal aggregate join is the theta join where the theta condition is applied to the reports' time intervals. In this section I focus on scenarios with no intersecting reports within a relation. The extension of these approaches to intersecting reports is left for future work. Nevertheless, these approaches can be used in case of intersecting reports if we first apply the RS method developed in [112].

5.4.6.1 The Goal of the Aggregate Join Given two relations r and s that are defined over schemas $R = (From, To, V_1)$ and $S = (From, To, V_2)$ respectively and can be either valid-time (VT) or aggregate-time (AT) relations, the goal is to obtain a relation j with schema $G = (From, To, V_1, V_2)$ by joining those tuples that can be joined directly based on the condition on time intervals and relying on user to resolve all non trivial situations (we will see what those are later in the text).

In some cases, one or the other variable is very important and the exact values of such variable are required whereas estimated values of the other variable will suffice. To express such cases when fusing two relations, left or right (or full if values of both variables are important) outer versions of joins can be used. Thus, the inner, left, right and full outer join should conform to the following properties:

- Inner Join – output only matching tuples without the use of any user functions.
- Left Outer Join – output matching tuples and also not matching tuples from the left relation.
    - Preserve the left variable (after normalization/zoom out) unmodified and only apply user functions to the variables of the right relation.
- Right Outer Join – symmetrically opposite definition of the left outer join.
- Full Outer Join – Union (with duplication removal) of Left and Right Outer Joins.

Explanations of what normalization, zoom out and user defined functions will be provided further in the text.

For the further discussion of the aggregate join methods, consider two relations r and s as shown in Figure 71. Notice that I don't specify what type (VT or AT) each relation is since it will vary while explaining different methods.

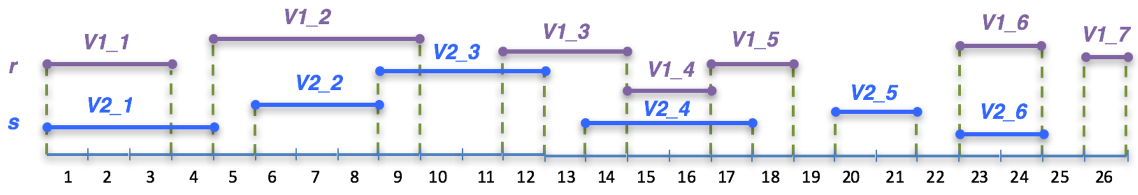


Figure 71: Example relations r and s for discussion of approaches for the aggregate join

5.4.6.2 Equi-join – The Baseline Method Regardless of the type of the relation, as the base method, consider an equi-join, $r \bowtie_{From,To} s$, of the two relations r and s shown in Figure 72. As stated in Definition 2, the equi-join doesn't take into consideration anything except the $From$ and To time endpoints of the two relation and only joins those reports that have matching time intervals.

$r \bowtie_{From,To} s$

From	To	V1	V2
23	24	V1_6	V2_6

Figure 72: The result of the equi-join, $r \bowtie_{From,To} s$, of the two relations shown in Figure 71

The left/right and full outer joins are executed in a straightforward way according to Definition 3 and their results are presented in Figure 73 and Figure 74 respectively.

$r ⟕_{From,To} s$

From	To	V1	V2
1	3	V1_1	ω
5	9	V1_2	ω
12	14	V1_3	ω
15	16	V1_4	ω
17	18	V1_5	ω
23	24	V1_6	V2_6
26	26	V1_7	ω

$r ⟖_{From,To} s$

From	To	V1	V2
1	4	ω	V2_1
6	8	ω	V2_2
9	12	ω	V2_3
14	17	ω	V2_4
20	21	ω	V2_5
23	24	V1_6	V2_6

Figure 73: The result of the left outer join, $r ⟕_{From,To} s$, and right outer join, $r ⟖_{From,To} s$

$r ⟗_{From,To} s$

From	To	V1	V2
1	3	V1_1	ω
1	4	ω	V2_1
5	9	V1_2	ω
6	8	ω	V2_2
9	12	ω	V2_3
12	14	V1_3	ω
14	17	ω	V2_4
15	16	V1_4	ω
17	18	V1_5	ω
20	21	ω	V2_5
23	24	V1_6	V2_6
26	26	V1_7	ω

Figure 74: The result of full outer equi-join, $r ⟗_{From,To} s$

As you can see from Figure 72 the result of the inner join only covers small portion of time units out of the lifespan of the two relations r and s. Thus, while the result is accurate, the coverage is very small. While left or right outer joins (Figure 73) preserve the complete coverage for r or

s relations respectively, most of the information of the other variable is not available and replaced with unknown values (ω). The result of the full outer join (Figure 74) has many intersecting time intervals making it hard to make any sense of the join result.

In what follows, I present several other approaches to perform aggregate join that behave differently depending on the type of the relation. The approaches are also different from the ones considered in the previous sections in that they don't try to estimate values in an uncertain situations and instead involve users to resolve them.

5.4.6.3 Temporal Alignment Join – Joining VT with VT or AT Relation In this section I focus on joining a valid time relation with either another valid time or with an aggregate time relation. The approach is similar to the Align join discussed in [53], however all the work in temporal database area only consider joining valid time relations.

Let us start with the first case when both relations r and s are valid time relations, i.e., r^{VT} and s^{VT}. Recall from Section 5.4.1 that a valid time relation describes the time interval when a tuple is true. Thus, the report $s_1^{VT} = (1, 4, V2\_1)$ means that at each time unit from 1 to 4 the value of V_2 is $V2\_1$. Therefore, we can split the report s_1^{VT} into two reports: $s_{1\_1}^{VT} = (1, 3, V2\_1)$ and $s_{1\_2}^{VT} = (4, 4, V2\_1)$ without loss in accuracy of the reports. Notice that the original $r_1^{VT} = (1, 3, V1\_1)$ report was not matching with s_1^{VT} on time interval (Figure 75 left plot), however now r_1^{VT} can be joined with $s_{1\_1}^{VT}$ (Figure 75 right plot). The process of splitting valid time reports is called $normalization$ and it is defined similar to the one in [53]. Below I provide its definition and explanation.

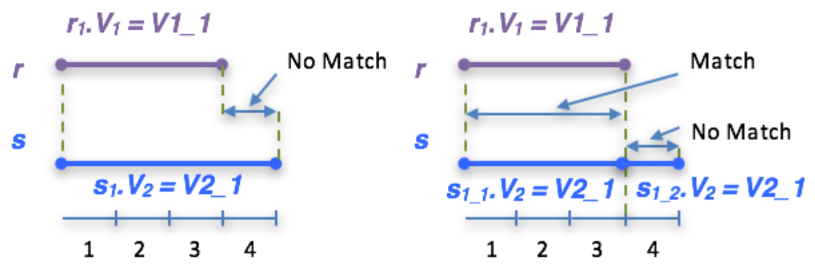


Figure 75: Example of normalization of the report s_1^{VT} over the report r_1^{VT}

Before defining the $normalization$ operator, I first define a helper operator, $temporal\ splitter$, similarly to [53] that splits time interval of one valid time report based on the time interval endpoints of the other report (can either be valid time or aggregate time type report).

Definition 11 – Temporal Splitter: The temporal splitter, $split(r^{VT}, s)$, of a valid time report r^{VT} and a set of either valid time or aggregate time reports s defined over schemas $R^{VT} = (From,\ To,\ V_1)$ and $S = (From,\ To, V_2)$ respectively, is defined as follows.

$$
\begin{aligned}
split(r^{VT}, s) = \{z^{(2)} \mid\ & \\
& z[T] \subseteq r^{VT}[T] \wedge \forall y \in s(\neg intersect(z[T], y[T]) \vee z[T] \subseteq y[T]) \wedge \\
& \forall T' \supset z[T](\exists y \in s\ (intersect(T', y[T]) \wedge (T' \nsubseteq r^{VT}[T] \vee T' \nsubseteq y[T])) \\
& \qquad \vee T' \nsubseteq r^{VT}[T])\}
\end{aligned}
$$

The second line of the definition requires that the new time interval, $z[T]$, is contained in the time interval of the report that is being split, $r^{VT}[T]$, and it is either not intersecting with time intervals of reports in s or it is contained in one of them. The third and fourth lines ensure that the time interval $z[T]$ is maximal.

□

Figure 76 shows five examples of applying the $temporal\ splitter$ operator. The resulting set z contains new time intervals of split r^{VT}.

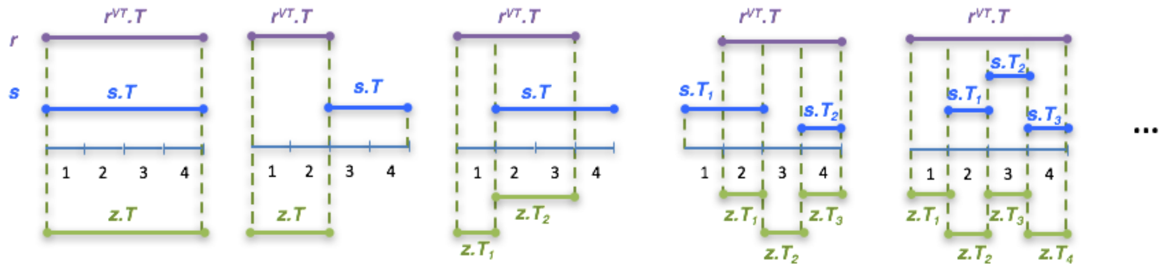


Figure 76: Example of normalization of a valid time report r^{VT} over reports in s

Definition 12 – Normalization of VT relation: The temporal $normalization$, $\mathcal{N}(r^{VT}, s)$, of a valid time relation over either a valid time or aggregate time relation defined over schemas $R^{VT} = (From,\ To, V_1)$ and $S = (From,\ To, V_2)$ respectively, is defined as follows.

$$\mathcal{N}(r^{VT}, s) = \{z^{(|R^{VT}|)} \mid \exists x \in r^{VT}\ ($$

$$z[V_1] = x[V_1] \wedge z[T] \in split(x,\ \{y \in s | intersect(x[T], y[T])\}))\}$$

The second line assigns the values of variables of the report that is being normalized and ensures that the time interval of a new normalized report is one of time intervals resulting from splitting the original report.

□

Note that our definition of normalization is slightly different from the temporal splitter operator proposed in [53] and normalization function of Toman [179] in that it does not consider any condition on explicit attributes and is extended to aggregate type relation. Figure 77 shows normalized relation r^{VT} over s^{VT}, $\mathcal{N}(r^{VT}, s^{VT})$, and s^{VT} over r^{VT}, $\mathcal{N}(s^{VT}, r^{VT})$, from Figure 71.

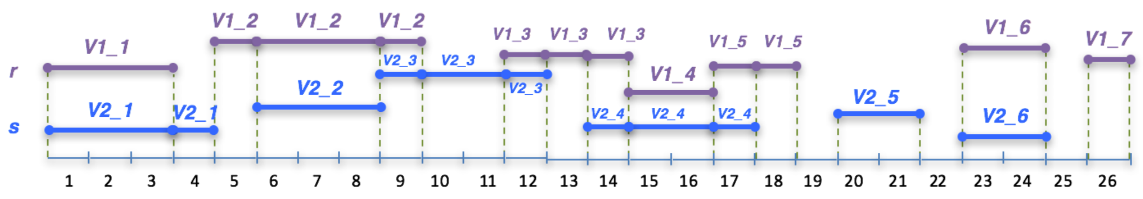


Figure 77: Normalized relation r^{VT} over s^{VT}, $\mathcal{N}(r^{VT}, s^{VT})$, and s^{VT} over r^{VT}, $\mathcal{N}(s^{VT}, r^{VT})$, from Figure 71

Definition 13 – Inner Temporal Alignment (TA) Join of two VT relations: The inner temporal alignment join, $r^{VT} \bowtie^{TA} s^{VT}$, of two valid time relation r^{VT} and s^{VT} defined over schemas $R^{VT} = (From,\ To, V_1)$ and $S = (From,\ To, V_2)$ respectively, is defined as follows.

$$r^{VT} \bowtie^{TA} s^{VT} = \mathcal{N}(r^{VT}, s^{VT}) \bowtie_{From,To} \mathcal{N}(s^{VT}, r^{VT})$$

□

The left, right and full outer temporal alignment joins are defined similarly by first normalizing both relations and then using the standard left, right, outer operators. Figure 78 shows the result of the temporal alignment inner join on the two relations r^{VT} and s^{VT} shown in Figure 71. Note that the temporal alignment join of two VT relations is exactly equivalent to the temporal Cartesian product defined in Definition 7. The difference of Temporal Alignment join compared to temporal Cartesian product will be present in the case of joining VT and AT relations.

Now, consider the case when the s relation is an aggregate time relation, s^{AT}. In such case, we cannot normalize s^{AT} over the r^{VT}, however we still can normalize r^{VT} over s^{AT}. Figure 79 shows the result of the normalization of r^{VT} over s^{AT}, $\mathcal{N}(r^{VT}, s^{AT})$.

$r^{VT} \bowtie^{TA} s^{VT}$

From	To	V1	V2
1	3	V1_1	V2_1
6	8	V1_2	V2_2
9	9	V1_2	V2_3
12	12	V1_3	V2_3
14	14	V1_3	V2_4
15	16	V1_4	V2_4
17	17	V1_5	V2_4
23	24	V1_6	V2_6

Figure 78: The result of the temporal alignment join, $r^{VT} \bowtie^{TA} s^{VT}$, of the two valid time relations shown in Figure 71

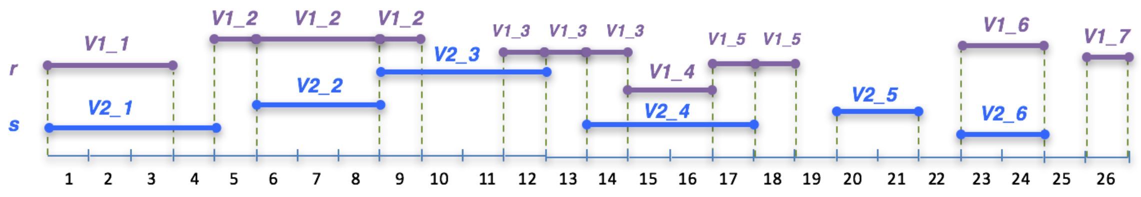


Figure 79: Normalized relation r^{VT} over s^{AT}, $\mathcal{N}(r^{VT}, s^{AT})$, from Figure 71

Note that if we perform the join of r^{VT} and s^{AT} as $r^{VT} \bowtie^{TA} s^{AT} = \mathcal{N}(r^{VT}, s^{AT}) \bowtie_{From,To} s^{AT}$, the join result might not cover some time units for which the exact value could be obtained. For example, consider time units from 14 to 17 on the Figure 79, the report $(14, 17, V2\_4)$ from s^{AT} doesn't match exactly with any of the reports $(14, 14, V1\_3)$, $(15, 16, V1\_4)$ and $(17,17, V1\_5)$ from r^{VT}, however the values of the three reports could be aggregated by applying a weighted mean (wm) (14) function where the weight for each report value is the length of the report, i.e., we can create a new report

$(14, 17, mean(V1\_3, V1\_4, V1\_4, V1\_5))$ that could be joined successfully with the report $(14, 17, V2\_4)$. Note that for the illustration purposes the value $V1\_4$ appears twice to show the fact that it covers two time units and thus need to be counted twice. I call the operation of merging several valid time reports as $stitching$ and I will define it below. On the other hand, notice that reports $(9, 9, V1\_2)$ and $(12, 12, V1\_3)$ cannot be stitched together due to the presence of a gap from time unit 10 to 11, and thus the report $(9, 12, V2\_3)$ cannot be joined with them. Note that in the further text I will talk about user defined fusion functions that could be applied to reports $(9, 9, V1\_2)$ and $(12, 12, V1\_3)$, however those functions are only applicable to the left, right and outer joins because otherwise they violate the property of the inner join defined in Section 5.4.6.1.

Definition 14 – Stitching: Given a normalized valid time relation r^{NVT} and an aggregate time relation s^{AT} defined over schemas $R^{VT} = (From,\ To, V_1)$ and $S = (From,\ To, V_2)$ respectively, the stitching of r^{NVT} over s^{AT}, $stitch(r^{NVT}, s^{AT})$, is defined as follows.

$$stitch(r^{NVT}, s^{AT}) = \{z^{(|R^{VT}|)}|$$

$$\exists y \in s^{AT}(\mathcal{B} = \{x | x \in r^{NVT} \wedge intersect(x[T], y[T])\} \wedge$$

$$\Big((card(\mathcal{B}) = 1\ \vee gaps(\mathcal{B})\big) \Rightarrow \forall x \in \mathcal{B}(z = x)\Big) \vee$$

$$(z[T] = lifespan(\mathcal{B}) \wedge z[V_1] = wm(\mathcal{B}))) \vee$$

$$\exists x \in r^{NVT} \forall y \in s^{AT}(\neg intersect(x[T], y[T]) \wedge z = x)\}$$

The lines two to four handle the case when a report from s^{AT} intersects with one or more report from r^{NVT} . If there is only one intersecting report or there are gaps between intersecting report then those reports are unmodified (line 3), otherwise the new stitched report will have lifespan of all the intersecting reports and the value will be equal to the result of applying the weighted mean function (14). The fifth line handles the reports from r^{NVT} that do no intersect with any report from s^{AT} and simply copies to the result. $card(\mathcal{B})$ is the cardinality of the relation $\mathcal{B}$.□

The weighted mean function is defined as follows:

$$wm(r^T) = \frac{\sum_{i=1}^{n}(r_i.V * |r_i|)}{\sum_{i=1}^{n}|r_i|}, \tag{14}$$

where n is the number of reports in r^T.

Figure 80 shows the result of stitching r^{NVT} over s^{AT}. As explained above, the three reports $(14, 14, V1\_3)$, $(15, 16, V1\_4)$ and $(17,17, V1\_5)$ from r^{NVT} relation were stitched together into one $(14, 17, wm(V1\_3, V1\_4, V1\_5))$.

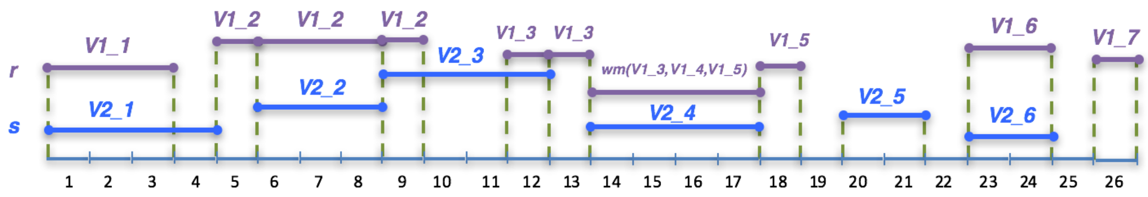


Figure 80: The result of stitching r^{NVT} over s^{AT}, $stitch(r^{NVT}, s^{AT})$

Now, we are ready to define a temporal alignment join of a valid time relation with an aggregate time relation.

Definition 15 – Inner Temporal Alignment (TA) Join of a VT and AT relations: The inner temporal alignment join, $r^{VT} \bowtie^{TA} s^{AT}$, of a valid time relation r^{VT} and an aggregate time relation s^{AT} defined over schemas $R^{VT} = (From,\ To, V_1)$ and $S^{AT} = (From,\ To, V_2)$ respectively, is defined as follows.

$$r^{VT} \bowtie^{TA} s^{AT} = stitch(\mathcal{N}(r^{VT}, s^{AT}), s^{AT}) \bowtie_{From,To} s^{AT}$$

□

$r^{VT} \bowtie^{TA} s^{AT}$

From	**To**	**V1**	**V2**
6	8	V1_2	V2_2
14	17	$wm(V1\_3, V1\_4, V1\_5)$	V2_4
23	24	V1_6	V2_6

Figure 81: The result of the temporal alignment join, $\mathrm{r}^{\mathrm{VT}} \bowtie^{\mathrm{TA}} \mathrm{s}^{\mathrm{AT}}$, of the valid time relation and the aggregate time relation shown in Figure 71

As you can see the cardinality (and therefore coverage) of the join shown in Figure 81 is smaller than in Figure 78, however it is better than blindly using standard equi join that would return only one tuple in Figure 72.

While for the outer joins we can use the standard approach, where for the not matching reports a null value is assigned, our goal when fusing datasets is to preserve as much information as possible. At the same time, as the experiments in Section 5.4.5.4 show, the best technique to fuse reports strongly depends on the nature of the data and the task at hand. Thus, instead of putting null values for each non-matching report or guessing how to fuse non-matching reports, I involve user in the loop by letting them to define what should be done in such non trivial situation by the means of $user\ defined\ functions$.

Let us look at an example of doing left outer temporal alignment join of a valid time relation r^{VT} and an aggregate time relation s^{AT}, $r^{VT}⟕^{TA}s^{\mathrm{AT}}$, shown in Figure 71. As with the inner join, we first do the normalization and then the stitching of the valid time relation r^{VT} obtaining the relations as in the Figure 80. Now, the goal of the left outer join is to preserve the left relation without any changes while either merging matching report or assigning null for not matching tuples in the right relation. Since the very first report $(1, 3, V1\_1)$ in r^{VT} doesn't match with any report in $s^{\mathrm{A}T}$, we would normally output a report like this: $(1, 3, V1\_1, \omega)$. However, we do have some information about $s^{\mathrm{A}T}$ over the time interval 1 to 3. We know the report $(1, 4, V1\_2)$, but since it is an aggregate time report, we don't know what are the actual values for each time unit or what is the aggregated value from time unit 1 to 3. So, we don't want to guess for the user what to do with it and let user handle such uncertain and non-trivial situation.

There are three basic types of user defined functions – $reduce$ (r), $combine$ (co) and $extend$ (e) – that aim to handle different scenarios. The illustrations of each function and their combinations into more complex functions are shown in Figure 82. The $reduce$ function is used when an aggregate time report has the time interval that extends the time interval of interest. The time interval of value $v21$ is longer then the time interval of interest of the $v11$. The $extend$ function is opposite from reduce and is used when the time interval needs to be extended. E.g., the time interval of $v21$ is shorter than $v11$. The $combine$ function is used when two or more reports need to be combined into one.

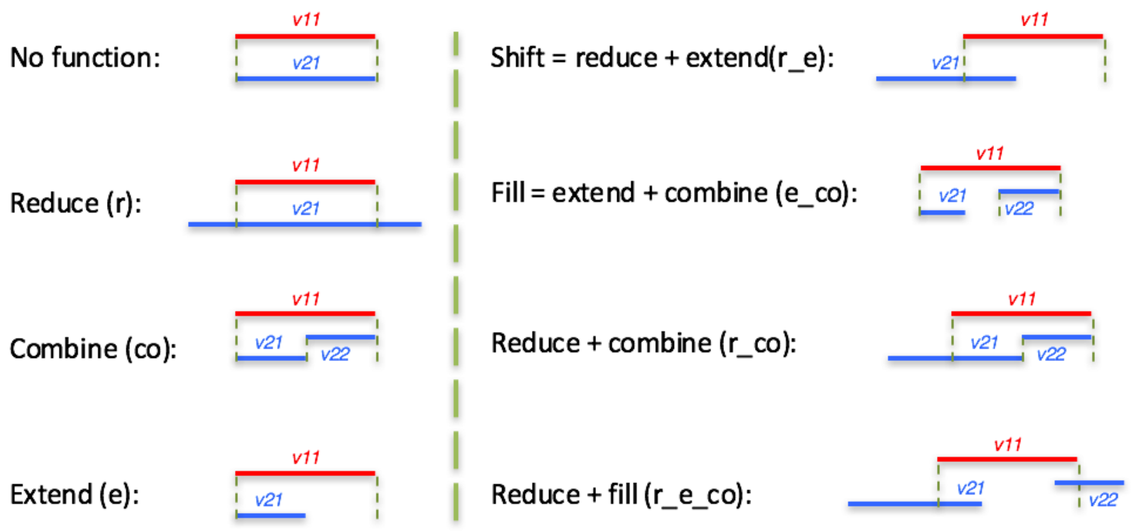


Figure 82: Illustration of types of user defined functions and their combinations to handle different cases of mutual position of several reports; red reports are the ones whose time interval we are interested to be unmodified; blue reports are those for which the user functions will be applied

The user functions can be either selected from a list of predefined general case functions (Table 7) or be an arbitrarily complex one. In addition to the values of the reports in question, user functions can take more information into account to compute the value. Such information can include for example other reports in the relations and relations' metadata. The concept of user defined functions to handle the non trivial situations is similar to the concept of conflict resolution functions in the work by Bleiholder and Neumann [24]. Table 7 shows a list of useful predefined user functions to merge non trivial reports that are based on the conflict resolution functions in Section 4 in [24] but are adapted to our use cases.

Table 7: Predefined user functions to merge non trivial reports

Function	Description
$Constant$	Return the provided constant value regardless of the values of the reports in question.
$Count$	Counts the number of distinct non-null values. The actual data values are lost.
Min/Max	Returns the minimal/maximal input value with its obvious meaning for numerical data. Lexicographical (or other) order is needed for non numerical data.
$Sum/Avg/Median$	Computes sum, average and median of all present non-null data values. Only applicable to numerical data. Takes into account TU form of the VT relations.
$Variance\ /\ Stddev$	Returns variance and standard deviation of data values. Only applicable to numerical data. Takes into account TU form of the VT relations.
$Random$	Randomly chooses one data value among all non-null data values.
$Choose$	Returns the value which satisfies provided conditions, e.g., min/max constraints, the report length, the source where it comes, etc.
$Coalesce$	Takes the first non-null value appearing.
$First/Last$	Takes the first/last value of all values, even if it is a null value.
$Vote$	Returns the value that appears most often among the present values. Ties can be broken by a variety of strategies, e.g., choosing randomly.
$Group$	Returns a set of all the values without performing any computation on them.
$Choose\ Corresponding$	Chooses the value that belongs to the value chosen for another column.
$Disaggregate$	Applies a specified disaggregation strategy to break intersecting reports and then combines finer granular values into one with the help of aggregate functions.

Taking the above into consideration, we now can define the left/right outer temporal alignment join. As the left and right outer joins are symmetric, we define only the left outer join.

Definition 16 – Left Temporal Alignment (TA) Join of a VT and AT relations: The left temporal alignment join, $r^{VT} ⟕^{TA} s^{AT}$, of a valid time relation r^{VT} and an aggregate time relation s^{AT} defined over schemas $R^{VT} = (From,\ To, V_1)$ and $S^{AT} = (From,\ To, V_2)$ respectively, is defined as follows.

$$r^{SNVT} = stitch(\mathcal{N}(r^{VT}, s^{AT}), s^{AT})$$

$$j = r^{SNVT} \bowtie_{From,To} s^{AT}$$

$$\theta = (r^{SNVT}.From > s^{AT}.To \wedge r^{SNVT}.From < s^{AT}.From) \vee$$
$$(r^{SNVT}.To > s^{AT}.To \wedge r^{SNVT}.To < s^{AT}.From)$$

$$i = {}_{r^{SNVT}.*}G_{uf(V_2)}(r^{SNVT} \bowtie_{\theta} s^{AT})$$

$$r^{VT} ⟕^{TA} s^{AT} = j \cup i \cup \pi_{*,\omega}(r^{SNVT} - \pi_{r^{SNVT}.*}(j) - \pi_{r^{SNVT}.*}(i))$$

The first line derives stitched normalized relation of r^{VT} over s^{AT}. The second line derives the inner temporal alignment join. The third and fourth lines define the theta condition, which finds intersecting (but not equal) reports between r^{SNVT} and s^{AT}. The fifth line performs grouping of the result of the theta join based on the all attributes of the r^{SNVT} relation by applying user functions (uf, combined from the three base function according to the Figure 82) to the V_2 values of each group. The sixth line performs the union of the inner join (j) with the reports that have intersections (i) with the reports form the left relation that were not matched during the inner or intersect joins by also appending the null value (ω) to each such report.

□

Figure 83 shows the result of the left and right temporal alignment outer joins.

$r^{VT} ⟕^{TA} s^{AT}$

From	To	V1	V2
1	3	V1_1	$r(V2\_1)$
5	5	V1_2	ω
6	8	V1_2	V2_2
9	9	V1_2	$r(V2\_3)$
12	12	V1_3	$r(V2\_3)$
13	13	V1_3	ω
14	17	$wm(V1\_3, V1\_4, V1\_5)$	V2_4
18	18	V1_5	ω
23	24	V1_6	V2_6
26	26	V1_7	ω

$r^{VT} ⟖^{TA} s^{AT}$

From	To	V1	V2
1	4	$e(V1\_1)$	V2_1
6	8	V1_2	V2_2
9	12	$e\_co(V1\_2, V1\_3)$	V2_3
14	17	$wm(V1\_3, V1\_4, V1\_5)$	V2_4
20	21	ω	V2_5
23	24	V1_6	V2_6

Figure 83: The result of the left temporal alignment outer join, $r^{VT} ⟕^{TA} s^{AT}$, and right temporal alignment outer join, $r^{VT} ⟖^{TA} s^{AT}$, of the two relation shown in Figure 71

The full outer temporal alignment join is defined as the standard outer join which is the union of the left and right temporal alignment joins. Figure 84 shows the result of the full outer temporal alignment join, $r^{VT} ⟗^{TA} s^{AT}$, of the valid time relation and the aggregate time relation shown in Figure 71. Notice that the full outer join contains some reports that intersect in time (the groups of intersecting report are highlighted in color). The resolution of such non-trivial situation is not straightforward and is left to the user to decide, but some heuristics can be applied. For example, the intersecting reports could be broken into finer granularity common time intervals, e.g., leave the report $(1, 3, V1\_1, r(V2\_1))$ without any changes, but change the second report to $(4, 4, e(V1\_1), r(V2\_1))$. Another heuristic could accumulate reports into coarser granularity, e.g., merge first two reports into $(1, 4, e(V1_1), V2\_1)$. Yet another heuristic could aim to minimize the number of user defined function. The work to develop such heuristics is left for future research.

$r^{VT} \bowtie^{TA} s^{AT}$

From	To	V1	V2
1	3	V1_1	$r(V2\_1)$
1	4	$e(V1\_1)$	V2_1
5	5	V1_2	ω
6	8	V1_2	V2_2
9	9	V1_2	$r(V2\_3)$
9	12	$e\_co(V1\_2, V1\_3)$	V2_3
12	12	V1_3	$r(V2\_3)$
13	13	V1_3	ω
14	17	$wm(V1\_3, V1\_4, V1\_5)$	ω
18	18	V1_5	ω
20	21	ω	V2_5
23	24	V1_6	V2_6
26	26	V1_7	ω

Figure 84: The result of the temporal alignment join, $r^{VT} \bowtie^{TA} s^{AT}$, of the valid time relation and the aggregate time relation shown in Figure 71

5.4.6.4 Overlap Join – Joining Two AT Relations In the case of joining two aggregate time relations, the normalization cannot be applied to any of them. Thus, the inner join of two aggregate time relations is performed as the standard intersect join. The intersect join in the context of valid time relations has been studied previously (e.g., in [52][64][124][103][102]) and efficient distributed algorithms are know (e.g., [52][105]). Below I provide approaches to perform join in the context of aggregate time relations. The result of the inner join of the two aggregate time relations from Figure 71 is shown in Figure 85.

$$\theta = (r^{AT}.From \geq s^{AT}.To \wedge r^{AT}.From \leq s^{AT}.From) \vee$$

$$(r^{AT}.To \geq s^{AT}.To \wedge r^{AT}.To \leq s^{AT}.From)$$

$$r^{AT} \bowtie_{\theta} s^{AT}$$

$r^{AT}.From$	$r^{AT}.To$	$s^{AT}.From$	$s^{AT}.To$	V1	V2
1	3	1	4	V1_1	V2_1
5	9	6	8	V1_2	V2_2
5	9	9	12	V1_2	V2_3
12	14	9	12	V1_3	V2_3
12	14	14	17	V1_3	V2_4
15	16	14	17	V1_4	V2_4
17	18	14	17	V1_5	V2_4
23	24	23	24	V1_6	V2_6

Figure 85: The result of the intersect join, $r^{AT} \bowtie_{\theta} s^{AT}$, of the two aggregate time relations shown in Figure 71

While the result of the inner overlap join doesn't directly return the required schema, i.e., $(From, To, V_1, V_2)$, it can be converted into it with another user defined function t that is applied to overlapping time interval within one report (Figure 86).

From	To	V1	V2
1	$t(3, 4)$	V1_1	V2_1
$t(5,6)$	$t(9,8)$	V1_2	V2_2
$t(5,9)$	$t(9,12)$	V1_2	V2_3
$t(12,9)$	$t(14,12)$	V1_3	V2_3
$t(12,14)$	$t(14,17)$	V1_3	V2_4
$t(15,14)$	$t(16,17)$	V1_4	V2_4
$t(17,14)$	$t(18,17)$	V1_5	V2_4
23	24	V1_6	V2_6

Figure 86: The result of applying t user function to the inner join in Figure 85

The overlap join merges reports that intersect in their time intervals without taking into consideration the length of the reports and their relative position and therefore the result of the join might contain tuples that will introduce overestimation or underestimation. Notice, for example, how in the Figure 71 the report $(9, 12, V2\_3)$ covers only 1 out of 5 time units of the report $(5, 9, V1\_2)$ (which is 20%), thus joining those two reports and then somehow fusing the $(9, 12, V2\_3)$ report with $(6, 8, V2\_2)$ may overestimate/underestimate the value of V_2 that corresponds to V_1 on that time interval. On the other hand, the reports that do not intersect but are very close to each other might provide good approximate join answer. For example, assume there is another report $(27, 27, V2\_7)$. It doesn't not intersect with $(26, 26, V1\_7)$, however since they are both very short (each only cover one time unit) and are very close (they are adjacent), joining them might provide good approximate for corresponding values of V_1 and V_2 on time interval 26 – 27. The intersect join will not identify such cases. To better illustrate the above problems, consider a scenario of reports shown in Figure 87. In the next two subsections I will describe two alternative techniques to perform intersect-like join.

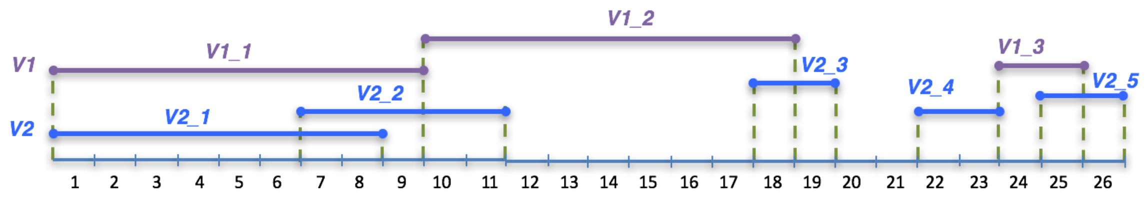


Figure 87: A scenario of reports to illustrate the problem with intersect join

5.4.6.4.1 Relative Overlap Join To address the first problem with intersect join approach identified above regarding joining the reports that have very short overlapping time intervals, a relative overlap metric, similarly to [190], can be used to judge whether the reports should be merged or not. The relative overlap (16) of two reports is the ratio of their overlap (7) over their combined lifespan which is the length of the union on their time intervals (15).

$$union(T_1, T_2) = [\min(T_1.From, T_2.From), \max(T_1.To, T_2.To)] \quad (15)$$

$$RO(T_1, T_2) = \frac{overlap(T_1, T_2)}{|union(T_1, T_2)|} \quad (16)$$

Figure 88 illustrates the relative overlap metric of two reports r_1 and s_1.

$$|r_1| = 7; \ |s_1| = 5$$

$$overlap(r_1.T, s_1.T) = 2$$

$$union(r_1.T, s_1.T) = 10$$

$$RO(r_1.T, s_1.T) = 0.2$$

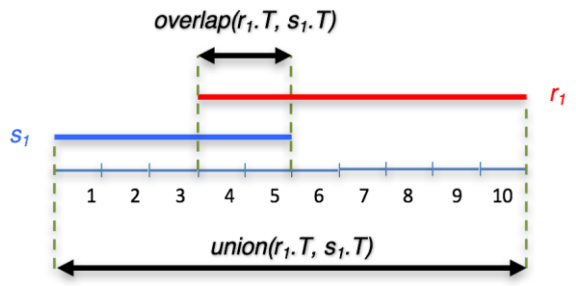


Figure 88: Illustration of relative overlap metric

Table 8 shows the non zero relative overlap values for the reports shown in Figure 87. The higher the value the better. As it can be seen from the figure, the reports that have longer overlapping time intervals have higher relative overlap value. The reports that don't intersect, have zero relative overlapping value (not shown in the table).

Table 8: Non zero relative overlap values for the reports in Figure 87

r.From	**r.To**	**s.From**	**s.To**	**RO**
1	9	1	8	0.88
1	9	7	11	0.27
10	18	7	11	0.16

Table 8 (continued)

10	18	18	19	0.1
24	25	25	26	0.33

Using the relative overlap metric and a given minimum relative overlap threshold, the relative overlap join can be defined as a theta join with predicate $a\ \theta\ b \equiv RO(a,b) \geq threshold$. A stricter version of the relative overlap join could be used to merge only those reports that have maximum relative overlap, then $a\ \theta\ b \equiv RO(a,b) = maximum$. Figure 89 shows the result of the relative overlap join with relative threshold 0.3 for the reports in Figure 87.

$$r^{AT} \bowtie_{RO(r^{AT},\, s^{AT}) \geq 0.3} s^{AT}$$

From	**To**	**V_1**	**From**	**To**	**V_2**
1	9	V1_1	1	8	V2_1
24	25	V1_3	25	26	V2_5

Figure 89: Relative overlap join query with threshold 0.3 and the result of the query for the aggregate time relations in Figure 87

5.4.6.4.2 Distance-based Join While the relative overlap join filters out reports that don't overlap "enough" based on the overlap threshold, it doesn't address the second issue identified with the intersect join. Namely, sometimes joining adjacent (or close "enough") reports might be useful to answer a join query approximately. The distance-based join method is conceptually similar to the idea of neighborhood based algorithms. Namely, this method matches a report of one relation with reports from the other relation that are in the neighborhood proximity based on the interval's start and end time values. This approach is similar to some degree to the work done by Pilourdault et al. [145] and by Dubois et al. [60] in which they extended Allen's algebra to account for approximate temporal predicates and to assign a score of equality and inequality of two time intervals' endpoints.

To calculate report's neighborhood, a distance measure is required. I use a distance measure (17), called $ti\_hausdorff$, that measures the distance between two time intervals. The measure is inspired by the Hausdorff distance that measures how far two sets are from each other.

$$ti\_hausdorff(T_1, T_2) = \max(d_1, d_2), \tag{17}$$

where

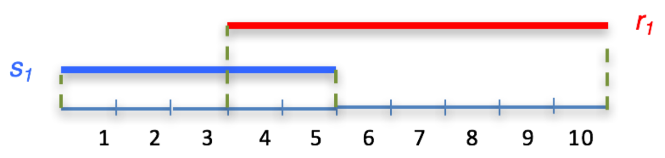
$$\begin{aligned} d_1(T_1, T_2) = \max(&\\ &\min\big(abs(T_1.From - T_2.From), abs(T_1.From - T_2.To)\big),\\ &\min\big(abs(T_1.To - T_2.From), abs(T_1.To - T_2.To)\big)\\ &)\end{aligned}$$

$$\begin{aligned} d_2(T_1, T_2) = \max(&\\ &\min\big(abs(T_2.From - T_1.From), abs(T_2.From - T_1.To)\big),\\ &\min\big(abs(T_2.To - T_1.From), abs(T_2.To - T_1.To)\big)\\ &)\end{aligned}$$

The abs function returns the absolute value of its argument.

Figure 90 illustrates the $ti\_hausdorff$ metric of two reports r_1 and s_1.

The $ti\_hausdorff$ distance returns the number of time units between two reports' time intervals and, thus, the value is not normalized between 0 and 1. To use the same semantics of the aggregate join as for the relative overlap, a normalized distance is used and it is calculated according to the following equation 18.



$$d_1 = \max(\min(4-1, abs(4-5)), \min(10-1, 10-5)) = 5$$

$$d_2 = \max(\min(abs(1-4), abs(1-10)), \min(5-4, abs(5-10))) = 3$$

$$ti\_hausdorff = 5$$

Figure 90: Illustration of $ti\_hausdorff$ metric

$$ti\_hausdorffN(T_1, T_2) = \frac{ti\_hausdorff(T_1, T_2)}{max\_ti\_hausdorff(r^{AT}, s^{AT})}, \quad (18)$$

where

$$max\_ti\_hausdorff(r^{AT}, s^{AT}) = z \iff$$

$$\forall x \in r^{AT} \forall y \in s^{AT} (z \geq ti\_hausdorff(x.T, y.T))$$

Table 9 shows the $ti\_hausdorffN$ values for the reports shown in Figure 87. The lower the value the better. As it can be seen from the figure, the reports that are further from each other have higher distance. The main difference with the relative overlap value is that $ti\_hausdorffN$ is not zero for non overlapping reports.

Table 9: Non zero $ti\_hausdorffN$ values for the reports in Figure 87

r.From	r.To	s.From	s.To	$ti\_hausdorffN$
1	9	1	8	0.04
1	9	7	11	0.25
1	9	18	19	0.71
1	9	22	23	0.87
1	9	25	26	1
10	18	1	8	0.42

Table 9 (continued)

10	18	7	11	0.29
10	18	18	19	0.33
10	18	22	26	0.5
10	18	25	26	0.62
24	25	1	8	0.95
24	25	7	11	0.71
24	25	18	19	0.25
24	25	22	23	0.08
24	25	25	26	0.04

Using the $ti\_hausdorffN$ metric and a given maximum $ti\_hausdorffN$ threshold, the distance-based join (I will refer to it also as hausdorff join) can be defined as a theta join with predicate $a\ \theta\ b \equiv ti\_hausdorffN(a, b) \leq threshold$. A stricter version of the hausdorff join could be used to merge only those reports that have minimum $ti\_hausdorffN$ value, then $a\ \theta\ b \equiv ti\_hausdorffN(a, b) = minimum$. Figure 91 shows the result of the hausdorff join with the threshold 0.2 for the reports in Figure 87. As you can see from the result of the join table, the main difference with the relative overlap join is that hausdorff join method also merges those tuples that don't intersect but are close "enough" (i.e., the distance is within the given threshold value).

$$r^{AT} \bowtie_{ti\_hausdorffN(r^{AT}, s^{AT}) \leq 0.2} s^{AT}$$

From	**To**	**V_1**	**From**	**To**	**V_2**
1	9	V1_1	1	8	V2_1
24	25	V1_3	22	23	V2_4
24	25	V1_3	25	26	V2_5

Figure 91: Hausdorff join query with threshold 0.2 and the result of the query for the aggregate time relations in Figure 87

Since the result of the relative overlap or hausdorff join depends on the threshold value, finding an optimal threshold value that maximizes the accuracy of the join is an interesting and important research question. I ran multiple experiments trying to identify optimal thresholds for different combination of values of many parameters (such as relation's lifespans, reports' lengths, the degree of reports intersections, the degree of scarcity of the reports and their relative position, noise and many other). The experiments showed that the threshold value is highly dependent on the nature of the data and the task at hand. Thus, I consider the task of selecting the appropriate threshold as resolving non trivial situations that should be implemented as user defined function.

In general, it is harder to answer inner query when both relations are of the aggregate time type. Left and right outer queries are more useful since they preserve at least one variable and thus can guarantee good performance for at least one variable.

The left outer overlap join is defined similarly to the left outer temporal alignment join in Definition 16 except that the normalization and stitching functions are not applied.

Definition 17 – Left Overlap Join of two AT relations: The left overlap join, $r^{AT} ⟕^{TO} s^{AT}$, of two aggregate time relations r^{AT} and s^{AT} defined over schemas $R^{AT} = (From,\ To, V_1)$ and $S^{AT} = (From,\ To, V_2)$ respectively, is defined as follows.

$$j = r^{AT} \bowtie_{From,To} s^{AT}$$

$$\theta = (r^{AT}.From > s^{AT}.To \wedge r^{AT}.From < s^{AT}.From) \vee$$
$$(r^{AT}.To > s^{AT}.To \wedge r^{AT}.To < s^{AT}.From)$$

$$i = {}_{r^{AT}.*}G_{uf(V_2)}(r^{AT} \bowtie_\theta s^{AT})$$

$$r^{AT} ⟕^{TO} s^{AT} = j \cup i \cup \pi_{*,\omega}(r^{AT} - \pi_{r^{AT}.*}(j) - \pi_{r^{AT}.*}(i))$$

□

The results of the left and right outer overlap joins are shown in Figure 92.

$r^{AT} ⟕^{TO} s^{AT}$

From	To	V1	V2
1	3	V1_1	$r(V2\_1)$
5	9	V1_2	$r\_e\_co(V2\_2, V2\_3)$
12	14	V1_3	$r\_e\_co(V2\_3, V2\_4)$
15	16	V1_4	$r(V2\_4)$
17	18	V1_5	$r\_e(V2\_4)$
23	24	V1_6	V2_6
26	26	V1_7	ω

$r^{AT} ⟖^{TO} s^{AT}$

From	To	V1	V2
1	4	$r(V1\_1)$	V2_1
6	8	$r(V1\_2)$	V2_2
9	12	$r\_e\_co(V1\_2, V1\_3)$	V2_3
14	17	$r\_co(V1\_3, V1\_4, V1\_5)$	V2_4
20	21	ω	V2_5
23	24	V1_6	V2_6

Figure 92: The result of the left outer overlap join, $r^{AT} ⟕^{TO} s^{AT}$, and right outer overlap join, $r^{AT} ⟖^{TO} s^{AT}$, of the two aggregate time relations shown in Figure 71

The full outer join is the standard union of the left and right joins. It has the same issue of having intersecting reports in the resulting table as in the case with full outer temporal alignment join of VT and AT relations. Same heuristics can be applied.

5.4.6.5 Zoom Out Join – Joining Two AT Relations Previous approach to join two aggregate time relations resulted in the use of user functions that might produce inaccurate estimated value. In this final method to perform aggregate join I focus on developing a method that can join two aggregate time relations to produce accurate values without any use of user functions in case of inner join.

In contrast to the temporal alignment join that breaks valid time reports into smaller pieces, I develop a *zoom out* join that combines several aggregate time reports together into one larger report before performing equi join.

Definition 18 – Inner Zoom Out Join of Two AT Relations: The $zoom\ out$ join, $r^{AT} \bowtie^{TZ} s^{AT}$, of two aggregate time relations r^{AT} and s^{AT} defined over schemas $R^{AT} = (From,\ To, V_1)$ and $S^{AT} = (From,\ To, V_2)$ respectively, is defined as follows.

$$c(x,y) = (first(x).From = first(y).From \wedge last(x).To = last(y).To \wedge$$

$$\neg gaps(x) \wedge \neg gaps(y))$$

$$r^{AT} \bowtie^{TZ} s^{AT} = \{z | \exists x \subseteq r^{AT} \exists y \subseteq s^{AT} ($$

$$\Big(c(x,y) \wedge \forall x' \supseteq x \forall y' \supseteq y\big(\neg c(x',y')\big)\Big) \Rightarrow$$

$$z.From = \ first(x).From \wedge z.To = last(x) \wedge$$

$$z.V_1 = af(x.V_1) \wedge z.V_2 = af(y.V_2))\}$$

The first two lines define a helper operator that given two sets of reports returns true if both sets match on their corner time endpoints and there are no gaps. The third and fourth lines find subsets of r^{AT} and s^{AT} that satisfy the helper operator and ensures that those reports are minimal. The last two lines generate a resulting tuple by setting the time interval equal to lifespan of the satisfied subsets and aggregating each variable by using the aggregate function (af) that was used to derive the original reports. In the case when the mean function was used originally, the weighted mean (wm) function (14) is used instead. □

Performing zoom out join on the two aggregate time relations as in the Figure 71 would yield an empty result because there are no subsets of r^{AT} and s^{AT} that satisfy the condition c. Consider two aggregate time relations shown in Figure 93. Notice that a subset of r^{AT} that consists of first two reports and a subset of s^{AT} that also consists of the first two reports satisfy the time interval constrains for the condition c. However the subset of s^{AT} has a gap at time unit 5 and thus these two subsets cannot be "zoomed out" and joined together. Another subset x of r^{AT} that consists of three reports with values $V1\_3, V1\_4$ and $V1\_5$ and the subset y of s^{AT} that consists of two reports with values $V2\_3$ and $V2\_4$ satisfy the c constraint and also the zoom out join condition and thus can be "zoomed out" and joined. Notice, that adding adjacent reports $V1\_6$ and $V2\_5$ to the x and y will still satisfy the c, but will violate the zoom out join constraint that requires the subsets to be minimal. The "zoomed out" versions of the relations r^{AT} and s^{AT} are shown in Figure 94 and the result of the inner zoom out join of them is shown in Figure 95.

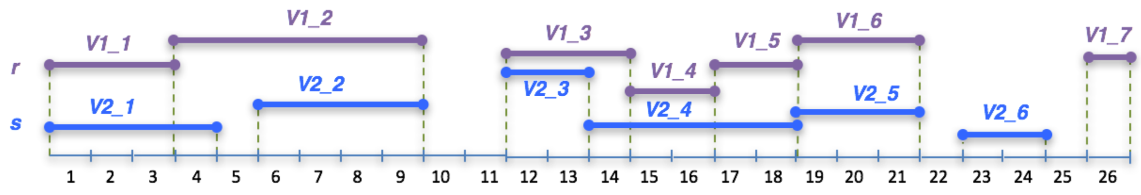


Figure 93: Example of two aggregate time relations for the illustration of the zoom out join approach

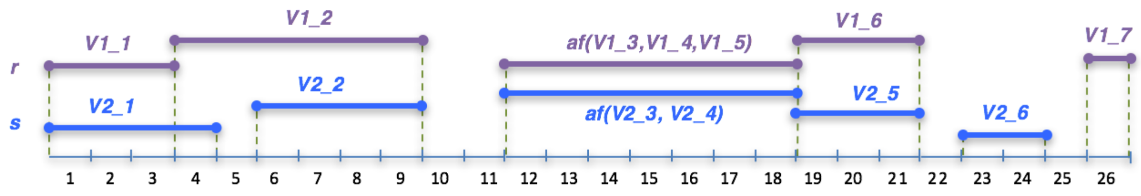


Figure 94: "Zoomed out" version of the two relations from Figure 93

$r^{AT} \bowtie^{TZ} s^{AT}$

From	To	V1	V2
12	18	$af(V1\_3, V1\_4, V1\_5)$	$af(V2\_3, V2\_4)$
19	21	V1_6	V2_5

Figure 95: Inner zoom out join, $r^{AT} \bowtie^{TZ} s^{AT}$, of the two aggregate relations from Figure 93

The semantics of the outer zoom out join method are the same as for the temporal alignment method.

Definition 19 – Left Zoom Out Join of two AT relations: The left zoom out join, $r^{AT} ⟕^{TZ} s^{AT}$, of two aggregate time relations r^{AT} and s^{AT} defined over schemas $R^{AT} = (From,\ To, V_1)$ and $S^{AT} = (From,\ To, V_2)$ respectively, is defined as follows.

$$j = r^{AT} \bowtie^{TZ} s^{AT}$$

$$lnj = r^{AT} - \pi_{r^{AT}.*}(j)$$

$$\theta = (lnj.From > s^{AT}.To \wedge lnj.From < s^{AT}.From) \vee$$
$$(lnj.To > s^{AT}.To \wedge lnj.To < s^{AT}.From)$$

$$i = {}_{lnj.*}G_{uf(V_2)}(lnj \bowtie_{\theta} s^{AT})$$

$$r^{AT} ⟕^{TZ} s^{AT} = j \cup i \cup \pi_{*,\omega}\left(lnj - \pi_{lnj.*}(i)\right)$$

The first line performs inner zoom out join of two relations of r^{AT} and s^{AT}. The second line find the reports from the r^{AT} relation that were not joined in the previous step and assigned them to a temporary relation lnj. The third and fourth lines define the theta condition which finds intersecting (but not equal) reports between lnj and s^{AT}. The fifth line performs grouping of the result of the theta join based on the all attributes of the lnj relation by applying user functions (uf, combined from the three base function according to the Figure 82) to the V_2 values of each group. The sixth line performs the union of the inner join (j) with the reports that have intersections (i) with the reports form the left relation that were not matched during the inner or intersect joins by also appending the null value (ω) to each such report.

□

The results of the left, right and full outer zoom out join can be derived similarly to the outer joins of the temporal alignment join.

An interesting scenario for the zoom out join is when both relations r^{AT} and s^{AT} have no gaps, no matching reports but have the same lifespan, e.g., as in Figure 96. After the zoom out process each relation will be aggregated into just one report, as in Figure 97, and the result of the inner and outer joins will be just one report $(1, 26,\ af(V1\_1,\ V1\_2,\ V1\_3,\ V1\_4,\ V1\_5,\ V1\_6,\ V1\_7),\ af(V2\_1,\ V2\_2,\ V2\_3,\ V2\_4,\ V2\_5,\ V2\_6))$. The result will be accurate and coverage will be perfect, but the granularity of the result will be coarse. Other approaches to join the reports in Figure 96 will do the following. The equi-join will not find any corresponding values between the two relations – inner join will return empty table; outer joins will yield null values for all time intervals for the other variable. While overlap join will preserve the original granularity, all reports in the result will have user defined functions and thus the join will be more computationally expensive and the accuracy can be low depending on the nature of the data and the choice of the user functions. Polynomial or SAFE disaggregation based join method could be used instead. However, since there are no overlapping reports the SAFE can have very high estimation error. If all the reports in each relation are of the same length, then temporal disaggregation methods could be applied and depending on the availability of good indicators, the join result could have the finest granularity and the lowest estimation error.

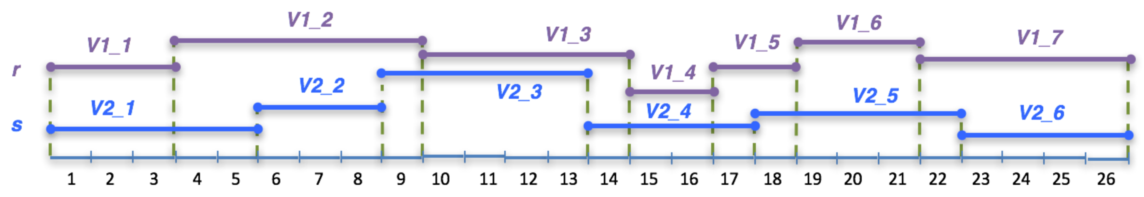


Figure 96: Example of an interesting scenario of two aggregate time relations for the zoom out join

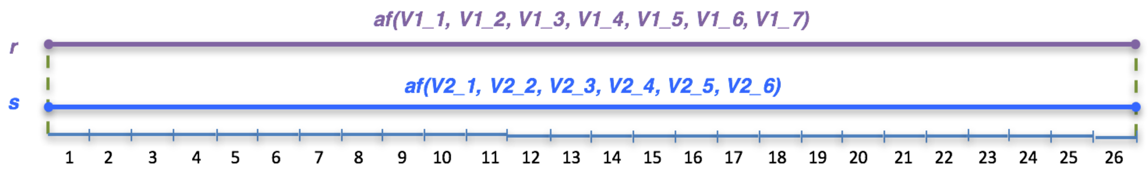

af(V1_1, V1_2, V1_3, V1_4, V1_5, V1_6, V1_7)
r
af(V2_1, V2_2, V2_3, V2_4, V2_5, V2_6)
s
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

Figure 97: Result of the zoom out join of the two aggregate time relations from the Figure 96

5.4.6.6 Implementations of Temporal Alignment and Zoom Out Aggregate Joins I implemented all join algorithms as sort-merge join. In fact, the relations are already sorted by time when they are given as input to a join algorithm and thus the join only need to perform the merge part by concatenating tuples that match. The implementations of the equi and overlap joins are straightforward since they do not require any additional operation to be performed on the joining relations. Below I provide algorithms to perform normalization (Algorithm 7), stitching (Algorithm 8) and zoom out (Algorithm 10). For these algorithms, I assume one-based arrays where first element has index 1 as opposite to the traditional zero-based array where first element has index 0.

In NORMALIZE algorithm (Algorithm 7), line 1 initiates an array of zeroes (ZEROS) on which lines 3 to 6 project the reports' time intervals from the relation s on the time axis t. Line 9 puts additional projects to the selected segment of t to indicate current reports' start (1) and end (end - last element of the t array) time points. Lines 7 to 17 break the relation that should be normalized r^{VT} into pieces based on the time interval endpoints projected on t.

For each aggregate time report, the STITCH algorithm (Algorithm 8) finds normalized valid time reports that are contained in it (line 14 to 22) and put those reports together by calling STITCHBAG (Algorithm 9). If there is a gap between VT reports within AT report (Algorithm 9, line 11), the gap is not filled up, but reports that form continuous strip are aggregated (line 12 and 20) by weighted mean function (wm (14)).

The ZOOMOUT algorithm (Algorithm 10) takes as input two aggregate time relations. It then first projects indexes of all reports of one relation on the time unit axis that those reports cover (line 1 to 4). The other report is then "zoomed out" by using ZOOMOUTONE algorithm (Algorithm 11) that takes into account both relations and the projects. The ZOOMOUTONE algorithm also outputs the list of time intervals that were zoomed out (the zoom out regions) that are then used by ZOOMOUTANOTHER algorithm (Algorithm 12) to zoom out the other relation.

Algorithm 7: NORMALIZE($\boldsymbol{r^{VT}}, \boldsymbol{s}$). **I**nput: Valid time relation $\boldsymbol{r^{VT}}$ that needs to be normalized, $\boldsymbol{s}$ either valid time or aggregate time relation based on which to do the normalization. **O**utput: Normalized relation $\boldsymbol{r^{NVT}}$.

```
16: $t \leftarrow$ ZEROS($1..\max(|r^{VT}|, |s|) + 1$)
17: $r^{NVT} \leftarrow$ empty list
18: **foreach** report $x$ of $s$ **do**
19:     $t[x.From] + +$
20:     $t[x.To + 1] + +$
21: **end foreach**
22: **foreach** report $y$ of $r^{VT}$ **do**
23:     $b = y.From; e = y.To; segment = t[b..e]$
24:     $segment[1] = 1; segment[end] = 1$
25:     $marks = \{ i \mid segment[i] > 0\}$
26:     **if** $|marks| > 2$ **then**
27:         **for** $i = 1; i < |marks|; i + +$ **do**
28:             $r^{NVT}$.APPEND($(b + marks[i] - 1, b + marks[i + 1] - 2, y.V)$)
29:         **end for**
30:     **else**
31:         $r^{NVT}$.APPEND($y$)
32:     **end if**
33: **end foreach**
34: return $r^{NVT}$
```

Algorithm 8: STITCH($\boldsymbol{r^{NVT}}$, $\boldsymbol{s}$). **I**nput: Normalized valid time relation $\boldsymbol{r^{NVT}}$ that needs to be stitched, $\boldsymbol{s}$ either valid time or aggregate time relation based on which to do the stitching. **O**utput: Normalized relation $\boldsymbol{r^{SNVT}}$.

```
1:  $r^{SNVT} \leftarrow$ empty list
2:  $i = 1$; $y = r^{NVT}[i]$; $rFinished = false$
3:  **foreach** report $x$ of $s$ **do**
4:      $bag \leftarrow$ empty list
5:      **while** $y.From < x.From$ **do**
6:          $r^{SNVT}$.APPEND($y$)
7:          $i++$
8:          **if** $i > card(r^{NVT})$ **then**
9:              $rFinished = true$
10:             **break**
11:         **end if**
12:         $y = r^{NVT}[i]$
13:     **end while**
14:     **while** $y.From \geq x.From$ **and** $y.To \leq x.To$ **do**
15:         $bag$.APPEND($y$)
16:         $i++$
17:         **if** $i > card(r^{NVT})$ **then**
18:             $rFinished = true$
19:             **break**
20:         **end if**
21:         $y = r^{NVT}[i]$
22:     **end while**
23:     $stitchedBag$ = STITCHBAG($bag$) //Algorithm 9
24:     $r^{SNVT}$.APPEND($stitchedBag$)
25:     **if** $rFinished$ **then**
26:         **break**
27:     **end if**
28: **end foreach**
29: **if** $i \leq card(r^{NVT})$ **then**
30:     $r^{SNVT}$.APPEND($\{r^{NVT}[j] | j \in [i..card(r^{NVT})]\}$)
31: **end if**
32: return $r^{SNVT}$
```

Algorithm 9: STITCHBAG($\boldsymbol{bag}$). **I**nput: A list of valid time reports that need to be stitched. **O**utput: A list of stitched reports $\boldsymbol{bag^S}$.

```
1:  $bag^S \leftarrow$ empty list
2:  $l =$ LENGTH($bag$)
3:  **if** $l == 0$ **then**
4:      return $bag^S$
5:  **end if**
6:  $x = bag[1]$
7:  $sBag \leftarrow empty\ list$
8:  $sBag.$ APPEND($x$)
9:  **if** $l > 1$ **then**
10:     **for** $i = 2; i \leq l$ **do**
11:         **if** $x.To + 1 \neq bag[i].From$ **then**
12:             $bag^S$.APPEND(($sBag[1].From, sBag[end].To, wm(sBag)$))
13:             $sBag \leftarrow empty\ list$
14:         **end if**
15:         $sBag$.APPEND($bag[i]$)
16:         $x = bag[i]$
17:     **end for**
18: **end if**
19: **if** LENGTH($sBag$) $> 0$ **then**
20:     $bag^S$.APPEND(($sBag[1].From, sBag[end].To, wm(sBag)$))
21: **end if**
22: return $bag^S$
```

Algorithm 10: ZOOMOUT($\boldsymbol{r^{AT}}, \boldsymbol{s^{AT}}$). **I**nput: Two aggregate time relations. **O**utput: Two zoomed out aggregated relations $\boldsymbol{r^{ZAT}}, \boldsymbol{s^{ZAT}}$.

```
1:  $t \leftarrow$ ZEROS($1..\max(|r^{AT}|, |s^{AT}|) + 1$)
2:  for $i = 1; i < card(s^{AT})$ do
3:      $t[s^{AT}[i].From..s^{AT}[i].To] = i$
4:  end for
5:  $(r^{ZAT}, zoomOutRegions) =$ ZOOMOUTONE($r^{AT}, s^{AT}, t$) // Algorithm 11
6:  $s^{ZAT} =$ ZOOMOUTANOTHER ($s^{AT}, zoomOutRegions$) // Algorithm 12
7:  return $r^{ZAT}$, $s^{ZAT}$
```

The ZOOMOUTONE algorithm (Algorithm 11) finds the minimum length continuous matching strips of reports in two relations and combines them together. Taking one report at a time if finds indexes of the reports of the other relation that lie under it (line 3). If there are gaps in other relation reports under current report (line 5), then the report is added to the output without any changes (line 6). If there is only one report in the other relation under current report and their time endpoints match (line 9), then the report is added to the output without any changes since this is already minimal zoom out (line 10). Otherwise, there is more than one report in the relation under the current report or those reports don't match and thus need to check for more conditions. If the reports match on the start endpoint and the next report is immediately adjacent (line 11), then the report is added to bag of potential candidates to be zoomed out (line 12). Otherwise, there is nothing can be done with the report, so it is added to the output (line 14). Lines 18 to 29 handle the case when the bag of potential candidates to zoom out is not empty and a decision for a new report needs to be made. First, the report is always added to the bag (line 18). However, if there are gaps in the other relation under current report, then the whole bag cannot be aggregated and thus all reports form it are added to the out put without any changes (line 20) and the bug is emptied (line 21). if current report matches the end time point of the underlying report in the other relation (line 22) then a minimum length continuous strip of reports in two relations is found and the bag can be aggregated, result added to the output and the bag is emptied (lines 23 - 25). The “zoomed out” time interval is also added to the zoomed out region list (line 24) that will be used by ZOOMOUTANOTHER algorithm (Algorithm 12). However, if the report end time point doesn’t match other relation and it is the last report or the next report is not adjacent (line 26), then nothing can be done and the report is added to the output without any changes (line 27) and the bag is emptied (line 28).

Algorithm 11: ZOOMOUTONE ($\boldsymbol{r^{AT}}, \boldsymbol{s^{AT}}, \boldsymbol{t}$). **I**nput: Two aggregate time relations and the projections of the indexes of the reports of the $\boldsymbol{r^{AT}}$ relation on the time axis. **O**utput: Zoomed out aggregated relation $\boldsymbol{r^{ZAT}}$ and the time intervals that were zoomed out.

```
1:  $r^{ZAT}$, $bag$, $zoomOutRegions \leftarrow empty\ list$;  $l = card(r^{AT})$
2:  for $i = 1; i < l; i++$ do
3:      $x = r^{AT}[i]$; $b = x.From$; $e = x.To$; $segment = t[b..e]$
4:      if LENGTH($sBag$) == 0 then
5:          if $\exists y \in segment(y == 0)$ then
6:              $r^{ZAT}$.APPEND($x$)
7:          else
8:              $uSeg$ = UNIQUE($segment$)
9:              if LENGTH($uSeg$) == 1 and $b == s^{AT}[uSeg[1]].From$ and $e == s^{AT}[uSeg[1]].To$ then
10:                 $r^{ZAT}$.APPEND($x$)
11:             elseif $b == s^{AT}[uSeg[1]].From$ and $i < l$ and $(r^{AT}[i+1].From - x.To == 1)$ then
12:                 $bag$.APPEND($x$)
13:             else
14:                 $r^{ZAT}$.APPEND($x$)
15:             end if
16:         end if
17:     else
18:         $bag$.APPEND($x$)
19:         if $\exists y \in segment(y == 0)$ then
20:             $r^{ZAT}$.APPEND($bag$)
21:             $bag \leftarrow empty\ list$
22:         elseif $e == s^{AT}[segment[end]].To$ then
23:             $r^{ZAT}$.APPEND(($bag[1].From, bag[end].To, af(bag)$))
24:             $zoomOutRegions$.APPEND($bag[1].From, bag[end].To$)
25:             $bag \leftarrow empty\ list$
26:         elseif $i == l$ or ($i < l$ and $(r^{AT}[i+1].From - x.To) > 1$) then
27:             $r^{ZAT}$.APPEND($bag$)
28:             $bag \leftarrow empty\ list$
29:         end if
30:     end if
31: end  for
32: return $r^{ZAT}$, $zoomOutRegions$
```

The ZOOMOUTANOTHER algorithm (Algorithm 12) takes in an aggregate time relation and zoom out time intervals. It iterates over all reports in the relation (lines 2 to 20 and 21 to 24), and the reports that fall in to the zoom out region are combined together (line 4 to 15) while other are added to the output without any changes (line 17 and 22).

Algorithm 12: ZOOMOUTANOTHER ($\boldsymbol{s^{AT}}, \boldsymbol{zoomOutRegions}$). **I**nput: An aggregate time relation and the zoom out time intervals. **O**utput: Zoomed out aggregated relation $\boldsymbol{s^{ZAT}}$.

```
1:  $s^{ZAT}$, $bag \leftarrow empty\ list$;   $i, j = 1$;   $l = card(s^{AT})$
2:  while $i \le l$ and $j \le$ LENGTH($zoomOutRegions$) do
3:      $x = s^{AT}[i]$
4:      if $x.From == zoomOutRegions[j].From$ then
5:          while $x.To \le zoomOutRegions[j].To$ do
6:              $bag$.APPEND($x$)
7:              $i{+}{+}$
8:              if $i > l$ then
9:                  break
10:             end if
11:             $x = s^{AT}[i]$
12:         end while
13:         $s^{ZAT}$.APPEND($bag[1].From, bag[end].To, af(bag)$)
14:         $bag \leftarrow empty\ list$
15:         $j{+}{+}$
16:     else
17:         $s^{ZAT}$.APPEND($x$)
18:         $i{+}{+}$
19:     end if
20: end while
21: while $i \le l$ do
22:     $s^{ZAT}$.APPEND($s^{AT}[i]$)
23:     $i{+}{+}$
24: end while
25: return $s^{ZAT}$
```

5.4.6.7 Empirical Evaluation In this section I describe the experimental evaluation of the aggregate join methods. If not said otherwise, the experiments were run on the Mac Book with Processor 2.4 GHz Intel Core i7 and 8Gb 1600 MHz DDR3 memory.

Since these methods do not try to estimate values on each time unit, there is no need to check for the accuracy of each method based on the errors between actual (ground truth) and estimated values. Instead, I focus on evaluating how aggregate join methods perform compare to the equi join method in terms of $cardinality$ of the joined table, its $coverage\ rate$, $granularity$ degree, $execution\ time$ and $fusion\ cost$.

The cardinality is the number of tuples (reports) in the join result. Since some of the tuples might have null values (w) or user defined functions, I calculate coverage of each time unit of a time interval according to the Table 10. If the user defined function is a combination of the basic functions, e.g. e_co, then the coverage values is equal to the minimum coverage value associated with each function, e.g., 0.4 because e has lower coverage than co. Coverage is calculated for each variable separately as the sum of each unit coverage. Coverage rate of one relation is the coverage of the relation in the join over the coverage of that relation in ground truth (original reports). Coverage rate of two relations is the average of coverage rates of two relations (19).

Table 10: Mapping table for coverage computation

Value	Number	$\boldsymbol{w}$	$\boldsymbol{t, e}$	$\boldsymbol{r}$	$\boldsymbol{co}$
Coverage	1	0	0.4	0.5	0.6

$$CR(l,r,j) = \frac{1}{2}\left(\frac{\sum_{i=1}^{z}|j_i|*coverage(j_i.V_1)}{\sum_{i=1}^{m}|l_i|*coverage(l_i.V_1)} + \frac{\sum_{i=1}^{z}|j_i|*coverage(j_i.V_2)}{\sum_{i=1}^{n}|r_i|*coverage(r_i.V_2)}\right), \tag{19}$$

where l, r are the left and right ground truth relations respectively, j is the join relation, n, m, z are the number of reports in l, r, j, $coverage(x)$ is the coverage of the value x form Table 10.

The granularity metric is calculated as the average report length in the relation according to the equation (20).

$$Granularity(r) = \frac{1}{n}\sum_{i=1}^{n}|r_i|, \tag{20}$$

Since we don't know how long each user defined function might take to compute, I calculate execution time of a join method as the elapsed time of executing the method without measuring time to also execute user functions. Thus, the execution time shows the time spent on additional functions that each methods does, e.g. normalization, stitching, zoom out, etc. I separately report what we call fusion cost that measures how much effort it would take the user to make sense of the join result. Fusion cost is equal to the number of user functions in the join result weighted by their corresponding cost according to Table 11. The value of the weight is not as important as the difference between them. We just want to show that resolving one report (e, r) is cheaper than two reports (co) which is itself cheaper than resolving time intervals (t).

Table 11: Mapping table for user defined function fusion cost

Function	$\boldsymbol{e, r}$	$\boldsymbol{co}$	$\boldsymbol{t}$
Cost	5	10	15

In this experiment, I measure how join methods perform when the size of the relations increases. Particularly, I vary the lifespan of each relation as 1000, 10000, 50000, 100000, 250000, 500000, and 1000000 time units. I repeat the experiment 5 times and report average values. For each lifespan case, I generate two relations each having number of reports up to the one tenth of the relation lifespan, i.e., 100, 1000, 5000, 10000, 25000, 50000, and 100000 reports respectively. Report lengths are drawn from a uniform distribution with minimum 10 and maximum 100 time units with step 10. Reports positions are also randomly generated from uniform distribution without intersections within one relation, but gaps are possible. Because of how the reports are generated, the actual average number of reports in each relation is much smaller. Average number of reports in left and right relations are shown in Table 12.

Table 12: Average number of reports in left and right relations for different lifespans

	1K	**10K**	**50K**	**100K**	**250K**	**500K**	**1M**
Left	22.0	200.4	943.4	1892.8	4633.4	9289.8	18413.2
Right	23.8	190.6	951.2	1863.6	4636.2	9253.4	18424.4

I execute the following queries:

- ejI, ejL, ejR – inner, left and right joins respectively using equi-join method.
- taI, taL, taR – inner, left and right joins respectively using temporal alignment join method.
- toI, toL, toR – inner, left and right joins respectively using overlap join method.
- tzI, tzL, tzR – inner, left and right joins respectively using zoom out join method.

The results of the experiment are shown below on Table 13, Table 14, Table 15, Table 18, Table 16 and Figure 98, Figure 99, Figure 100, Figure 102, Figure 101.

From the execution time perspective all methods show good scalability with largest joins not exceeding 8.5 seconds on average. However, for many methods the trend is exponential. As expected, equi-join shows fastest performance in terms of execution time (Table 13, Figure 98) and fusion cost (Table 14, Figure 99) (by definition equi join doesn't apply any user function). However as we can see from coverage (Table 15, Figure 100) or cardinality (Table 16, Figure 101) data, inner equi-join is unable to join many reports together.

All other methods show quite similar performance in terms of execution time. However, there is quite a big difference between them in terms of fusion cost (Table 14, Figure 99). As expected, taI and tzI have zero fusion cost since they do not apply any user functions, while the fusion cost of the toI join spiked due to the large number of overlapping aggregate time reports which cannot be fused automatically. The outer versions of joins show expected performance with noticeable different in the taL and taR cases. Since in the case of temporal alignment one relation is of valid time type, the algorithm knows how to reduce (by normalization) and then combine (by stitching) valid time reports and does not require r and co user functions to perform fusion.

Generated reports aligned in such a way so that zoom out join could not find many opportunities to perform the zoom out. In terms of coverage (Table 15, Figure 100) and cardinality (Table 16, Figure 101) its performance is close to the equi-join. However, from execution time perspective it is much slower than the equi-join since it tried to find potential reports to zoom out. Two other methods, temporal alignment and overlap, are able to achieve high coverage and cardinality, meaning that they were able to merge many tuples from the two relations. As expected, cardinality (Table 16, Figure 101) of toL, toR and taR are equal to their corresponding equi join versions (ejL, ejR) since these algorithms preserve left/right relations without any modifications. The case of taL join is different because left relation is a valid time and thus is normalized based on the right relation first. Therefore, the long original reports are split into several shorter reports and thus increase the cardinality of the join result. The high cardinality of the toI join is due to the same reason why toI has very high fusion cost. An interesting comparison can be made about cardinality of inner equi and zoom out joins and their left and right outer versions. Cardinality of inner zoom out join is larger than the cardinality of inner equi join because zoom out join can identify some reports that are not matching by themselves, but are matching if combined together into longer reports. In case of outer joins, equi join preserve all reports from the original left and right relations, whereas zoom out join combines some reports together and thus output smaller number of joined reports.

Table 17 shows average granularity of original left and right relations, and Table 18 and Figure 102 show the average granularity of the join results. Left and right outer joins for all methods (except taL) produce reports that are close in granularity to the original reports which is the expected behavior. Because of the normalization, taL and taI produce shorter reports while toI and tzI produce longer reports due to intersection and zoom out respectively.

Table 13: Average execution time in seconds of each query on different relation lifespan

	1K	**10K**	**50K**	**100K**	**250K**	**500K**	**1M**
ejI	0.0001	0.0005	0.0019	0.0037	0.0108	0.0184	0.0413
ejL	0.0002	0.0006	0.0022	0.0044	0.0097	0.0195	0.0462
ejR	0.0002	0.0006	0.0022	0.0044	0.0099	0.0196	0.0401
taI	0.0029	0.0206	0.0988	0.2040	0.5087	1.1847	3.0214
taL	0.0084	0.0757	0.3736	0.7673	1.8008	3.7895	8.3634
taR	0.0070	0.0581	0.2833	0.5757	1.3338	2.8967	6.2614
toI	0.0032	0.0304	0.1581	0.3865	0.8088	1.5203	2.7029
toL	0.0085	0.0757	0.3738	0.7442	1.6544	3.4666	6.8600
toR	0.0086	0.0758	0.3675	0.7244	1.6789	3.4076	6.7807
tzI	0.0018	0.0146	0.0658	0.1324	0.3054	0.6648	1.4793
tzL	0.0108	0.0893	0.4180	0.8517	1.9653	4.0915	8.0981
tzR	0.0100	0.0909	0.4247	0.8426	1.9562	4.0918	8.1929

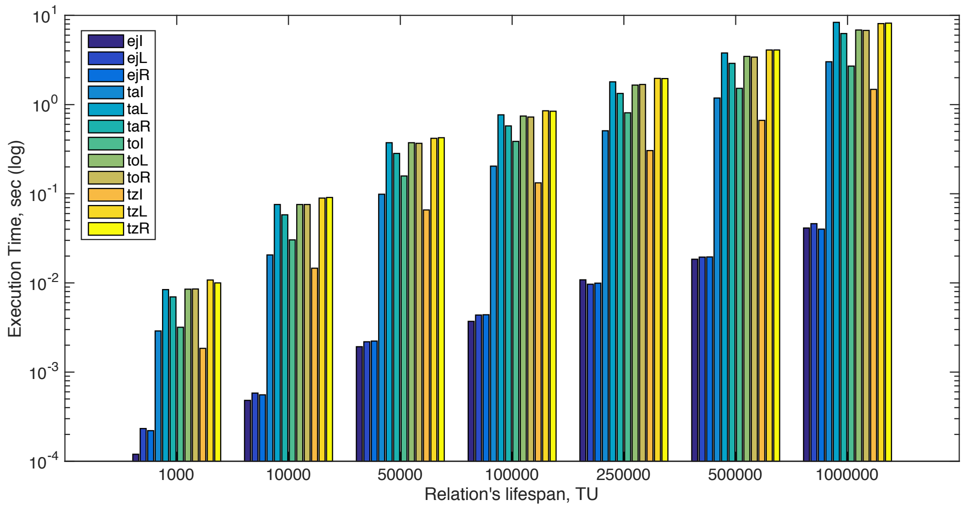


Figure 98: Average execution time in seconds (log scale) versus the lifespan of relations in time units

Table 14: Average fusion cost of each query on different relation lifespan

	1K	10K	50K	100K	250K	500K	1M
ejI	0	0	0	0	0	0	0
ejL	0	0	0	0	0	0	0
ejR	0	0	0	0	0	0	0
taI	0	0	0	0	0	0	0
taL	53	541	2680	5502	13603	27664	55281
taR	31	290	1444	2914	7220	14690	29291
toI	1281	11004	53790	107274	264255	528765	1050924
toL	165	1439	6839	13793	33995	68178	134557
toR	165	1379	6942	13709	33887	67924	134715
tzI	0	0	0	0	0	0	0
tzL	165	1422	6779	13741	33858	67794	133843
tzR	165	1362	6889	13659	33752	67574	134088

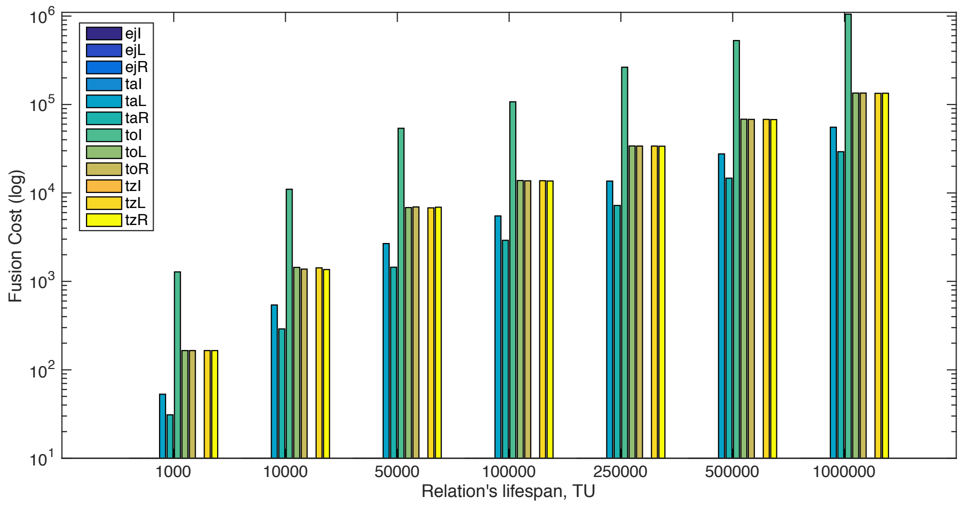


Figure 99: Average fusion cost (log scale) versus the lifespan of relations in time units

Table 15: Average coverage of each query on different relation lifespan

	1K	**10K**	**50K**	**100K**	**250K**	**500K**	**1M**
ejI	0.0000	0.0045	0.0040	0.0018	0.0014	0.0017	0.0021
ejL	0.5000	0.5023	0.5020	0.5009	0.5007	0.5008	0.5010
ejR	0.5000	0.5023	0.5020	0.5009	0.5007	0.5008	0.5010
taI	0.6608	0.6113	0.6169	0.6028	0.6064	0.5993	0.5985
taL	0.9080	0.8956	0.8970	0.8933	0.8943	0.8923	0.8922
taR	0.8957	0.8831	0.8850	0.8807	0.8818	0.8799	0.8795
toI	0.4144	0.4101	0.4102	0.4112	0.4112	0.4113	0.4111
toL	0.7357	0.7439	0.7471	0.7457	0.7444	0.7442	0.7464
toR	0.7401	0.7467	0.7457	0.7440	0.7453	0.7451	0.7459
tzI	0.0000	0.0198	0.0136	0.0074	0.0071	0.0087	0.0087
tzL	0.7357	0.7481	0.7496	0.7471	0.7457	0.7460	0.7480
tzR	0.7401	0.7509	0.7482	0.7455	0.7468	0.7469	0.7477

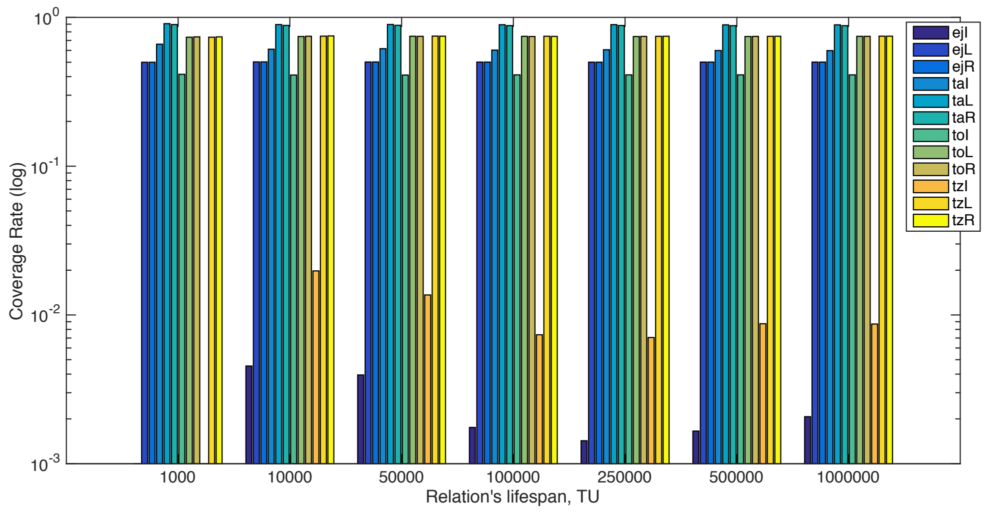


Figure 100: Average coverage rate versus the lifespan of relations in time units

Table 16: Average cardinality of each query on different relation lifespan

	1K	10K	50K	100K	250K	500K	1M
ejI	0.0	0.6	4.2	3.6	7.4	16.2	40.8
ejL	22.0	200.4	943.4	1892.8	4633.4	9289.8	18413.2
ejR	23.8	190.6	951.2	1863.6	4636.2	9253.4	18424.4
taI	17.6	132.6	662.4	1280.8	3192.2	6315.4	12566.2
taL	36.0	304.8	1512.6	3011.8	7481.0	14978.4	29908.6
taR	23.8	190.6	951.2	1863.6	4636.2	9253.4	18424.4
toI	43.4	371.6	1814.4	3607.6	8899.2	17803.4	35378.0
toL	22.0	200.4	943.4	1892.8	4633.4	9289.8	18413.2
toR	23.8	190.6	951.2	1863.6	4636.2	9253.4	18424.4
tzI	0.0	1.8	7.2	7.0	16.6	36.4	78.4
tzL	22.0	198.8	936.8	1887.0	4617.6	9247.6	18335.4
tzR	23.8	189.0	945.8	1858.4	4622.8	9216.2	18359.8

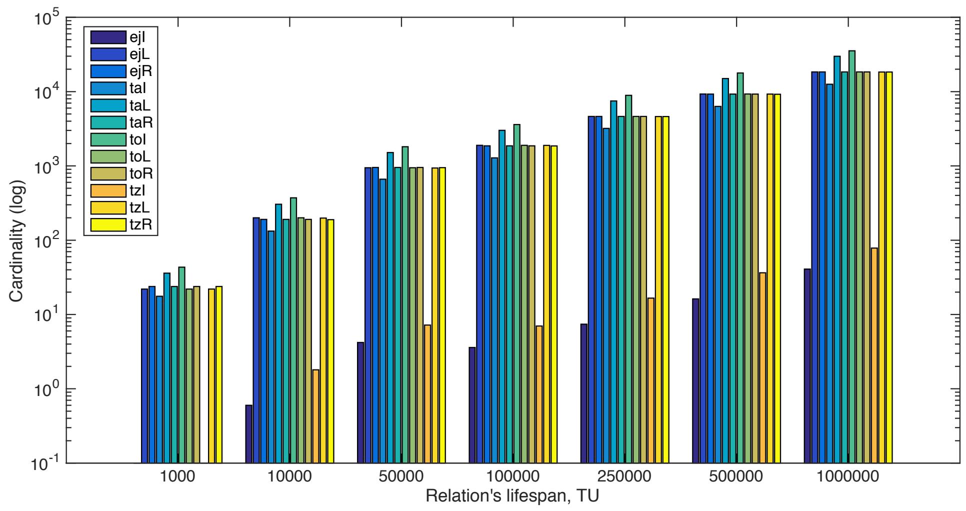


Figure 101: Average cardinality versus the lifespan of relations in time units

Table 17: Average granularity of left and right relations for different lifespans

	1K	10K	50K	100K	250K	500K	1M
Left	44.31	48.45	51.48	51.27	52.38	52.20	52.69
Right	40.35	50.90	51.03	52.05	52.33	52.42	52.66

Table 18: Average granularity of each query on different relation lifespan

	1K	10K	50K	100K	250K	500K	1M
ejI	0.00	44.00	43.06	41.50	46.98	49.70	48.53
ejL	44.31	48.45	51.48	51.27	52.38	52.20	52.69
ejR	40.35	50.90	51.03	52.05	52.33	52.42	52.66
taI	36.72	44.77	45.20	45.66	46.10	46.02	46.21
taL	26.93	31.89	32.12	32.22	32.44	32.37	32.44
taR	40.35	50.90	51.03	52.05	52.33	52.42	52.66
toI	80.81	92.32	93.25	93.87	94.66	94.47	94.91
toL	44.31	48.45	51.48	51.27	52.38	52.20	52.69
toR	40.35	50.90	51.03	52.05	52.33	52.42	52.66
tzI	0.00	91.67	76.65	98.67	100.55	116.55	107.13
tzL	44.31	48.84	51.84	51.43	52.56	52.44	52.91
tzR	40.35	51.34	51.33	52.20	52.48	52.64	52.84

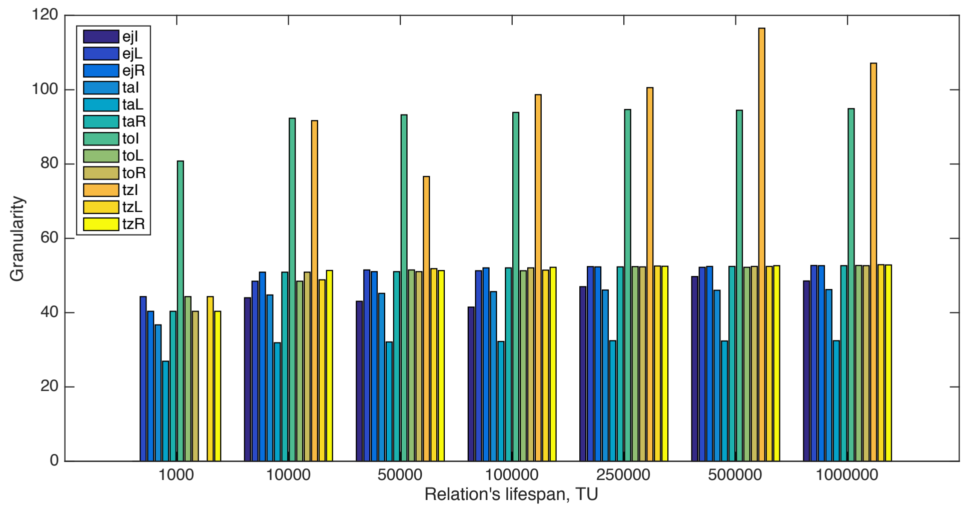


Figure 102: Average granularity versus the lifespan of relations in time units

Table 19 shows the overall comparison of the 12 join queries on all five metrics. The green color represents good quality, red means bad quality. Note that sometimes the "Low" value is bad (e.g., low coverage) and sometimes it is good (e.g., low execution time), the same for the "High" value. In case of the cardinality metric, for the taL query the "High" value is good because the temporal alignment method is able to find many matches without the use of many user functions (fusion cost is "Medium"), but in case of toI the "Extra high" value is not good, because there are many intersecting aggregate time reports that require "Extra high" fusion cost. For the granularity metric, the "Alike" value means that the granularity of the join reports is similar to the granularity of the original reports. The "Finer" value is better because then the join reports cover shorter time intervals and thus provide more information about the behavior of the variable. The "Coarser" is opposite, meaning that the join reports are longer than the original reports.

Table 19: Overall comparison of the 12 join queries on all five metrics

	Execution time	**Fusion cost**	**Coverage**	**Cardinality**	**Granularity**
ejI	Low	None	Low	Low	Alike
ejL	Low	None	Medium	Expected	Alike
ejR	Low	None	Medium	Expected	Alike
taI	Medium	None	Medium	Medium	Finer
taL	High	Medium	High	High	Finer
taR	High	Medium	High	Expected	Alike
toI	Medium	Extra High	Medium	Extra High	Coarser
toL	High	High	High	Expected	Alike
toR	High	High	High	Expected	Alike
tzI	Medium	None	Low	Low	Coarser
tzL	High	High	High	Expected	Alike
tzR	High	High	High	Expected	Alike

5.4.7 Temporal Join Conclusion

Figure 103 summarizes all the join approaches that I have explained in the previous sections. Given two temporal relations we can perform join of them in many ways. We can ignore the relations type (valid time or aggregate time) and blindly perform standard equi join or intersect join. While the standard approaches are the most straightforward to implement and use, they might result in lower cardinality and coverage of the join result or merge temporal intervals that are intersect only on few time units (or fail to merge closely related time intervals). The aggregate join approaches are suited for both valid time and aggregate time relations and depending on reports' relative position and lengths can produce the result with high cardinality and coverage while sometimes can sacrifice the granularity. Moreover, the aggregate join methods always return exact values except for the non trivial cases where user defined functions are applied. The disaggregate join approaches can be applied to the aggregate time relations to estimate values on finer granularity (high frequency) based on the coarse granular reports. The quality of the estimation depends on many parameters and thus accuracy of the join result cannot be guaranteed. In addition, these methods have many applicability limitations. As it commonly happens, there is no single "silver bullet" method that can be applied in any use case and show the best performance. Meanwhile, they can be combined in efficient information fusion framework that takes into account their complimentary applicability limitations.

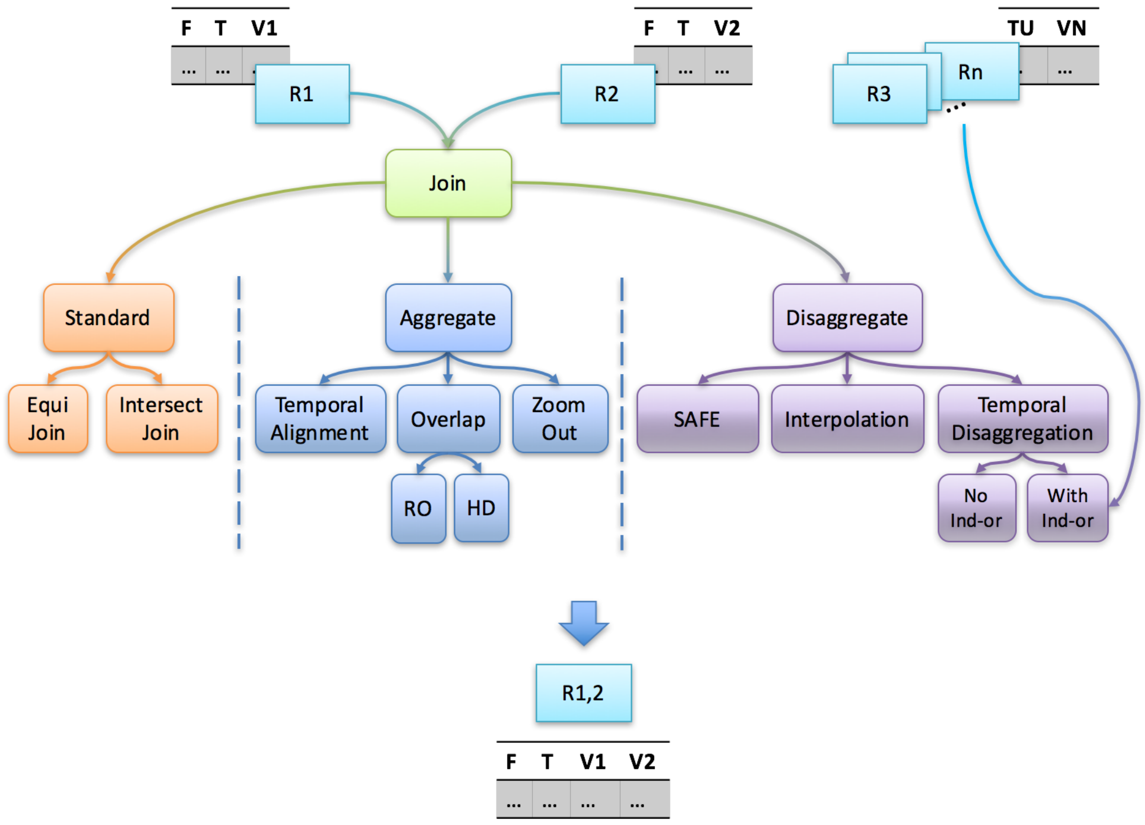


Figure 103: Illustration of join approaches. RO is the Relative Overlap method; HD is the Hausdorff method, SAFE is the spread, aggregate, fill, extend method; Interpolation is the method that is based on Polynomial interpolation or Spline; Ind-or means indicator

5.5 FUSE JOIN

Building on the information provided in previous sections we would like to have a join strategy that would intelligently provide the best effort to fuse datasets by applying the appropriate join methods. I call such join strategy a $fuse\ join$. Consider, for example, two relations $R = (name, location, time, value1)$ and $S = (name, location, time, value2)$. The name attribute represents some attribute of string domain, location attribute represents spatial domain (in our case it is also a string value, not geographic coordinates), time attribute represents temporal domain and the value attribute represents some numeric value of interest. We want to know the corresponding values from R and S, in other words we want to join R and S relations. The fuse join of the two relation can be written as in equation (21) and would be automatically translated into the join query that uses appropriate join methods as in equation (22).

$$R \bowtie^{FJ} S \tag{21}$$

$$R \bowtie^{T}_{sed(R_{name}, S_{name}) \leq nth \wedge ted\ (R_{location}, S_{location}) \leq lth} S, \tag{22}$$

where nth is the threshold for the string edit distance and lth is the threshold for the tree edit distance.

5.5.1 Empirical Evaluation

To show that fuse join strategy performs better than standard join methods, i.e., the equi join on all or some join attributes, I have conducted the following experiment (I repeated it for 5 times and report average values). First, I generated two relations r^{VT} and s^{AT} defined over schemas R and S as above, both having exactly the same values for name and location attributes, but different time intervals. The actual numeric value of the value attributes is not important at this moment. The location attribute has three levels with one distinct value on the first level, two distinct values on the second level and three distinct values on the third level. For each combination of name and location values a random number of time units between 3 and 20 was chosen from a uniform distribution and a random value for each time unit was generated. The values were then aggregated into reports with random lengths between 1 and half of the number of time units using 'no overlaps but gaps possible' strategy. Then I distorted 50% of the name and location values in the s^{AT} by introducing up to 3 random typos into name values and randomly removing up to 2 levels from location attribute.

Table 20 shows the seven queries that I executed. Q_1 joins the two relations only based on the equality of the values in the name attribute. Notice that due to the distortion that I performed on the s^{AT} relation, this query will not be able to join some tuples that in fact represent the same entity. Additionally, this query may introduce false positives by ignoring the location attribute completely. Q_2 joins the two relations based on the approximate string join on the $name$ attribute. While this query will be able to find more matching tuples that represent the same entity, it may introduce false positive by joining tuples whose names distance is smaller than the threshold but which represent different entities, and similarly to Q_1 it ignores the location attribute. Q_3 joins the two relation based on the equality of $name$ and $location$ values. While this query will find true matching entities, it will fail to match those tuples which have typos in name and different granularity level of location value. Since it does not take into account the time attribute, it may also join values that were reported at disjoin time intervals. Q_4 query uses string edit distance on the location attribute which may not be helpful since the location value could be spelled in completely different ways but still be close in the location hierarchy. Q_5 uses the tree edit distance on the location attribute, but still it ignores the time attribute and thus may have the same problems as the previous queries. Q_6 differs from Q_5 in the usage of the time attribute, but it joins only those tuples where time intervals match exactly, and thus may miss information on overlapping intervals. Finally, Q_7 represents the most comprehensive fuse join query that applies appropriate methods based on the type of the attribute. In particular, in contrast to Q_6, it uses temporal alignment join method to join valid time relation and aggregate time relation.

Table 20: Queries for the experiment on fuse join

Query Name	Query
$\boldsymbol{Q_1}$	$R^{VT} \bowtie_{name} S^{AT}$
$\boldsymbol{Q_2}$	$R^{VT} \bowtie_{sed(R_{name}, S_{name}) \le nth} S^{AT}$
$\boldsymbol{Q_3}$	$R^{VT} \bowtie_{name, location} S^{AT}$
$\boldsymbol{Q_4}$	$R^{VT} \bowtie_{sed(R_{name}, S_{name}) \le nth \wedge sed(R_{location}, S_{location}) \le nth} S^{AT}$
$\boldsymbol{Q_5}$	$R^{VT} \bowtie_{sed(R_{name}, S_{name}) \le nth \wedge ted(R_{location}, S_{location}) \le lth} S^{AT}$
$\boldsymbol{Q_6}$	$R^{VT} \bowtie_{sed(R_{name}, S_{name}) \le nth \wedge ted(R_{location}, S_{location}) \le lth \wedge time} S^{AT}$
$\boldsymbol{Q_7}$	$R^{VT} \bowtie^{TA}_{sed(R_{name}, S_{name}) \le nth \wedge ted(R_{location}, S_{location}) \le lth} S^{AT}$

To evaluate the result of each query, I use precision, recall and F$_1$ metrics calculated according to the equations (23), (24) and (25) respectively.

$$precision\ (P) = \frac{|TM \cap JM|}{|JM|} \tag{23}$$

$$recall\ (R) = \frac{|TM \cap JM|}{|TM|} \tag{24}$$

where TM is the set of truly matching tuples and JM is the set of tuples matched by a query.

$$F_1 = 2\frac{precision \cdot recall}{precision + recall} \tag{25}$$

Table 21 shows the result of the experiment. We can see that $Q_1 - Q_5$ have very low precision because they do not consider all attributes and thus join result contains some matches that were produced as a Cartesian product. The recall is also very low because the queries ignore the time component. Therefore, simply returning all possible combination of tuples will not give perfect recall like in the information retrieval field. As soon as we take time into consideration, Q_6 and Q_7, the precision increases significantly. However, the recall of Q_6 is very low, because it only joins those tuples where time intervals match exactly (which turned out to be a very low number). Since some queries use threshold for the string edit distance and/or for the tree edit distance, in general the quality of the join will depend on the choice of the threshold. As the right part of the table shows, if we increase the threshold values, the precision drops significantly due to the increased number of the false positive matches.

Table 21: Precision, Recall and F1 of the seven join queries

	$nth = 1, lth = 1$			$nth = 3, lth = 2$		
	P	R	F1	P	R	F1
Q_1	*0.003*	*0.070*	*0.006*	0.0034	0.0746	0.0066
Q_2	0.003	0.083	0.006	0.0004	0.0992	0.0009
Q_3	0.019	0.061	0.029	0.0212	0.0673	0.0322
Q_4	0.005	0.066	0.009	0.0004	0.0745	0.0009
Q_5	0.010	0.083	0.018	0.0006	0.0992	0.0013
Q_6	0.487	0.094	0.158	0.0354	0.1281	0.0554
Q_7	**0.461**	**0.643**	**0.537**	**0.0364**	**1.0000**	**0.0702**

6.0 DISSERTATION CONCLUSION AND FUTURE WORK

This dissertation summarizes my work towards the grand vision described in Section 1.2, particularly the part of the vision related to the data sharing/archiving, integration and fusion.

The task of data sharing and integration has been an active focus of research and development work in both academia and industry. In Chapter 2.0 I provide comprehensive review of existing approaches and point out their applicability limitations. I then, in Chapter 3.0, describe in detail the infrastructure that I proposed and developed. Particularly, in the chapter, I answer the question posed in Section 1.3. I developed distributed heterogeneous data storage and ingestion process (Section 3.1) to store dynamically incoming heterogeneous datasets efficiently by also enabling both data integration and data autonomy. Opposite to existing data repositories, the datasets in my infrastructure are semantically integrated (Section 3.2) by combination of machine learning algorithms and human expertise to perform efficient schema alignment and maintain relationships between the datasets. The semantic relationship model that virtually integrates distinct heterogeneous datasets turned out to be very useful for efficient data exploration without requiring users to write complex queries. Section 3.3 explains in details the model and algorithms that I developed to provide efficient key word functionality.

I have implemented the infrastructure in an easy to use rich web application (Section 4.0) that was tested and used for the historical data in tight collaboration with CHIA. From the experience of collaboration with CHIA and I've learned a number of lessons that needed to be addressed in order to make the infrastructure even more advanced. Particularly, the task of fusing datasets was the most interesting and challenging.

Chapter 5.0, that answers the question how to perform approximate information fusion when exact match does not exist, makes up a major chunk of my work and this document. I consider the task of data fusion as the ad-hoc relational database join query. While many algorithms were developed to perform string similarity joins and many specific index structures were developed to perform spatial joins, no attention has been given to task for joining relations that report aggregated values over some time intervals. The extensive and comprehensive studies in the area of temporal database management systems only focused on modeling valid time semantics in which temporal join algorithms can simply be rewritten as non temporal ones. In my work I raise the problem of joining the relations where the values are aggregated over time intervals and are not known at each time unit. I systemize the problem and provide high level framework to address it at different levels. Particularly, one way to join two aggregate time relation is first to estimate values for each time unit and then use standard join algorithm; while the other way is to join aggregated reports directly. Multiple experiments showed that the quality of the disaggregate join depends on the nature of the data and reports' characterizes described in Section 5.4.3. Therefore, I developed several algorithms to join aggregated reports. The algorithms do not perform any estimation and instead rely on user involvement in the join task to resolve all non trivial situations. Multiple experiments showed that the algorithms that I developed allow to execute join efficiently while preserving low fusion cost and high coverage. Finally, I showed that combination of approximate join techniques on string, spatial and temporal data allows to significantly improve precision and recall of the join result over the standard equi join methods.

There are several directions for the future work. On the storage level, similar to the BigDAWG Polystore System [61], data stores that support various data models, besides the relational one, need to developed and integrated into the infrastructure. More algorithms need to implemented to integrate various data types and to allow for efficient visual exploration of large datasets and relationships between them.

Particularly many research questions that need further investigation are related to the data fusion:

- Are some operations on the aggregated values isomorphic to the operations on the fine granular data and how do they depend on the nature of the data? For example, is the data analysis on the aggregate join result will produce the same results as on the fine granular (high frequency) data? If we must disaggregate the reports, should we disaggregate all them and to which granularity level? How to identify the best indicator series?
- If join algorithms depends on the nature of the data, then how can we keep and use some kind of index/metadata/statistics on what the nature of the data is. How to learn parameters to perform all the queries without user involvement (e.g. edit distance threshold, hausdorff threshold, choose user defined function for the temporal join, etc.)?
- Can we develop parametric characteristics (similar to statistics in database manage systems that influence which algorithm to use during planning/optimization phase) of dataset scenarios that would allows to quickly assess which join strategy to use.

- How to implement the fuse join in distributed environment and to scale it to large datasets? A special index might help speed up join process (e.g. like R trees for spatial data).

Some currently active and future development tasks are also explained in Section 4.2.

BIBLIOGRAPHY

[1] Aberer, K. 2011. Peer-to-Peer Data Data Management. *Synthesis lectures on data management*. 3, 2 (2011), 1–150.

[2] Aditya, B. 2002. BANKS : Browsing and Keyword Searching in Relational Databases. *Proceedings of the 28th international conference on Very Large Data Bases* (2002).

[3] Afrati, F.N., Sarma, A. Das, Menestrina, D., Parameswaran, A. and Ullman, J.D. 2012. Fuzzy joins using MapReduce. *Proceedings - International Conference on Data Engineering*. (2012), 498–509.

[4] Agarwal, S., Keller, A.M., Wiederhold, G. and Saraswat, K. 1995. Flexible relation: an approach for integrating data from multiple,possibly inconsistent databases. *Proceedings of the Eleventh International Conference on Data Engineering*. (1995).

[5] Agrawal, S., Chaudhuri, S. and Das, G. 2002. DBXplorer: a system for keyword-based search over relationaldatabases. *Proceedings 18th International Conference on Data Engineering*. (2002).

[6] Alexe, B., Cate, B. TEN, Kolaitis, P.G. and Tan, W.-C. 2011. Characterizing schema mappings via data examples. *ACM Transactions on Database Systems*. 36, 4 (Dec. 2011), 1–48.

[7] Allemang, D. and Hendler, J. 2011. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier.

[8] Allen, J.F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM*. 26, 11 (1983), 832–843.

[9] Apache Marmotta: *http://marmotta.apache.org/*. Accessed: 2015-03-28.

[10] Augsten, N. and Böhlen, M.H. 2013. Similarity Joins in Relational Database Systems. *Synthesis Lectures on Data Management*. 5, 5 (Nov. 2013), 1–124.

[11] Aumueller, D., Do, H.-H., Massmann, S. and Rahm, E. 2005. Schema and ontology matching with COMA++. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05* (New York, New York, USA, 2005), 906.

[12] B., A., M., R. and W.-C., T. 2014. Preference-aware integration of temporal data. *Proceedings of the VLDB Endowment*. 8, 4 (2014), 365–376.

[13] Baid, A., Rae, I., Li, J., Doan, A. and Naughton, J. 2010. Toward scalable keyword search over relational data. *Proceedings of the VLDB Endowment*. (2010).

[14] Barcellan, R. 2003. ECOTRIM: a program for temporal disaggregation of time series. *Workshop on Quarterly National Accounts, Eurostat, Theme* (2003), 79–95.

[15] Beckmann, N., Kriegel, H.-P., Schneider, R. and Seeger, B. 1990. *The R*-tree: an efficient and robust access method for points and rectangles*. ACM.

[16] Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A. and Hedeler, C. 2010. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. *EDBT '10 Proceedings of the 13th International Conference on Extending Database Technology*. (2010), 573–584.

[17] Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E. and Widom, J. 2009. Swoosh: a generic approach to entity resolution. *The VLDB Journal—The International Journal on Very Large Data Bases*. 18, 1 (2009), 255–276.

[18] Bernstein, P. and Melnik, S. 2007. Model management 2.0: manipulating richer mappings. *In Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. (2007), 1–12.

[19] Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L. and Zaihrayeu, L. 2002. Data Management for Peer-to-Peer Computing: A Vision. *Proc. of the 5th Int. Workshop on the Web and Databases, WebDB* (2002), 89–94.

[20] Bertossi, L. 2006. Consistent query answering in databases. *ACM SIGMOD Record*.

[21] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S. and Sudarshan, S. 2002. Keyword searching and browsing in databases using BANKS. *Proceedings 18th International Conference on Data Engineering*. (2002).

[22] Bizer, C., Heath, T. and Berners-Lee, T. 2009. Linked data-the story so far. *International journal on Semantic Web and Information Systems*. (2009).

[23] Bleiholder, J. and Naumann, F. 2008. Data fusion. *ACM Computing Surveys*.

[24] Bleiholder, J. and Naumann, F. 2005. Declarative data fusion - Syntax, semantics, and implementation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 3631 LNCS, (2005), 58–73.

[25] Bohannon, P., Fan, W., Flaster, M. and Rastogi, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05* (2005), 143.

[26] Bonifati, A., Chang, E.Q., Ho, T. and Lakshmanan, L.V.S. 2005. HepToX: Heterogeneous Peer to Peer XML Databases. (May 2005).

[27] Boot, J.C.G., Feibes, W. and Lisman, J.H.C. 1967. Further methods of derivation of quarterly figures from annual data. *Applied Statistics*. (1967), 65–75.

[28] Brodie, M.L. 2010. Data Integration at Scale: From Relational Data Integration to Information Ecosystems. *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*.

[29] Brown, I. 2012. An Empirical Comparison of Benchmarking Methods for Economic Stock Time Series. *US Census Bureau*. (2012).

[30] Bruno, N., Gravano, L. and Marian, A. 2002. Evaluating top-k queries over Web-accessible databases. *Proceedings 18th International Conference on Data Engineering*. (2002).

[31] Bry, F. 1997. Query answering in information systems with integrity constraints. *Integrity and Internal Control in Information Systems*. Springer. 113–130.

[32] Buneman, P., Tan, W. and Khanna, S. 2001. Why and Where : A Characterization of Data Provenance. *Database Theory—ICDT*. (2001), 316–330.

[33] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D. and Rosati, R. 2001. Data Integration in Data Warehousing. *International Journal of Cooperative Information Systems*. 10, 03 (Sep. 2001), 237–271.

[34] Carey, M.J., Haas, L.M., Schwarz, P.M., Arya, M., Cody, W.F., Fagin, R., Flickner, M., Luniewski, a. W., Niblack, W., Petkovic, D., Thomas, J., Williams, J.H. and Wimmers, E.L. 1995. Towards heterogeneous multimedia information systems: the Garlic approach. *Proceedings RIDE-DOM'95. Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*. (1995), 124–131.

[35] Casters, M., Bouman, R. and Van Dongen, J. 2010. *Pentaho Kettle solutions: building open source ETL solutions with Pentaho Data Integration*. John Wiley & Sons.

[36] Chamberlin, G. 2010. Temporal disaggregation. *Economic & Labour Market Review*. (2010), 106–121.

[37] Chaudhuri, S. and Dayal, U. 1997. An overview of data warehousing and OLAP technology. *ACM Sigmod record*. 26, 1 (1997), 65–74.

[38] Chawda, B., Gupta, H. and Negi, S. 2014. Processing Interval Joins On Map-Reduce. *Edbt*. (2014), 463–474.

[39] Chen, B. and others 2007. An empirical comparison of methods for temporal distribution and interpolation at the national accounts. *Bureau of Economic Analysis*. (2007).

[40] Chen, G., Hu, T., Jiang, D., Lu, P., Tan, K.-L., Vo, H.T. and Wu, S. 2012. BestPeer++: A Peer-to-Peer Based Large-Scale Data Processing Platform. *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. (Apr. 2012), 582–593.

[41] Chiang, Y.-H., Doan, A. and Naughton, J.F. 2014. Modeling entity evolution for temporal record matching. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), 1175–1186.

[42] Christen, P. 2012. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering*. 24, 9 (Sep. 2012), 1537–1555.

[43] Crosas, M. 2011. The dataverse network®: an open-source application for sharing, discovering and preserving data. *D-Lib Magazine*. (2011).

[44] Dagum, E.B. and Cholette, P.A. 2006. *Benchmarking, temporal distribution, and reconciliation methods for time series*. Springer Science & Business Media.

[45] Dallachiesa, M., Eldawy, A., Ilyas, I.F. and Tang, N. NADEEF : A Commodity Data Cleaning System Categories and Subject Descriptors. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* 541–552.

[46] Dash: *https://dash.cdlib.org/*. Accessed: 2015-05-04.

[47] Data Documentation Initiative (DDI): *http://www.ddialliance.org/*. Accessed: 2015-03-29.

[48] Databib: *http://databib.org/*.

[49] Date, C.J., Darwen, H. and Lorentzos, N. 2002. *Temporal data & the relational model*. Elsevier.

[50] Demartini, G., Difallah, D.E. and Cudré-Mauroux, P. 2013. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *The VLDB Journal*. 22, 5 (Jul. 2013), 665–687.

[51] Denton, F.T. 1971. Adjustment of monthly or quarterly series to annual totals: an approach based on quadratic minimization. *Journal of the American Statistical Association*. 66, 333 (1971), 99–102.

[52] Dignös, A., Böhlen, M.H. and Gamper, J. 2014. Overlap Interval Partition Join. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. (2014), 1459–1470.

[53] Dignös, A., Böhlen, M.H. and Gamper, J. 2012. Temporal Alignment. November (2012), 109–123.

[54] Ding, B., Yu, J.X., Wang, S., Qin, L., Zhang, X. and Lin, X. 2007. Finding top-k min-cost connected trees in databases. *Proceedings - International Conference on Data Engineering*. (2007), 836–845.

[55] Doan, A., Halevy, A. and Ives, Z. 2012. *Principles of Data Integration*. Elsevier.

[56] Doan, A. and Halevy, A.Y. 2005. Semantic integration research in the database community: A brief survey. *AI magazine*. 26, 1 (2005), 83.

[57] Dreyfus, S.E. and Wagner, R.A. 1971. The Steiner problem in graphs. *Networks*. 1, 3 (1971), 195–207.

[58] Dryad: *http://datadryad.org/*.

[59] Dublin Core: *http://dublincore.org/*. Accessed: 2015-03-29.

[60] Dubois, D., HadjAli, A. and Prade, H. 2003. Fuzziness and uncertainty in temporal reasoning. *J. UCS*. 9, 9 (2003), 1168.

[61] Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T. and Zdonik, S. 2015. The BigDAWG Polystore System. *ACM SIGMOD Record*. 44, 2 (2015), 11–16.

[62] Dyreson, C., Grandi, F., Käfer, W., Kline, N., Lorentzos, N., Mitsopoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B. and others 1994. A consensus glossary of temporal database concepts. *ACM Sigmod Record*. 23, 1 (1994), 52–64.

[63] Elmagarmid, A.K., Ipeirotis, P.G. and Verykios, V.S. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*. 19, 1 (2007), 1–16.

[64] Enderle, J., Hampel, M. and Seidl, T. 2004. Joining interval data in relational databases. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*. Sigmod (2004), 683.

[65] Etal, B. et al. 2013. Why linked data is not enough for scientists. *Future Generation Computer Systems*. 29, 2 (Feb. 2013), 599–611.

[66] Fernandez, R.B. 1981. A methodological note on the estimation of time series. *The Review of Economics and Statistics*. 63, 3 (1981), 471–476.

[67] Fielding, R.T. 2000. *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

[68] Fonzo, T. Di 1990. The Estimation of M Disaggregate Time Series when Contemporaneous and Temporal Aggregates are Known. *The Review of Economics and Statistics*. 72, 1 (1990),

178–182.

[69] Franklin, M., Halevy, A. and Maier, D. 2005. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*. December (2005), 1–7.

[70] Franklin, M., Kossmann, D., Kraska, T., Ramesh, S. and Xin, R. 2011. CrowdDB: answering queries with crowdsourcing. *SIGMOD '11: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. (2011).

[71] Gao, D., Jensen, C.S., Snodgrass, R.T. and Soo, M.D. 2005. Join operations in temporal databases. *The VLDB Journal*. 14, 1 (2005), 2–29.

[72] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y. and Ullman, J. 1997. The TSIMMIS approach to mediation: Data models and languages. *Journal of intelligent information systems*. 8, 2 (1997), 117–132.

[73] Gonzalez, H., Halevy, A. and Jensen, C. 2010. Google fusion tables: data management, integration and collaboration in the cloud. *Proceedings of the 1st ACM symposium on Cloud computing* (2010), 175–180.

[74] Gregory C. Chow, A.L. 1971. Best Linear Unbiased Interpolation, Distribution, and Extrapolation of Time Series by Related Series. *The Review of Economics and Statistics*. 53, 4 (1971), 372–375.

[75] Gu, L., Baxter, R., Vickers, D. and Rainsford, C. 2003. Record linkage: Current practice and future directions. *CSIRO Mathematical and Information Sciences Technical Report*. 3, (2003), 83.

[76] Guay, A. and Maurin, A. 2015. Disaggregation methods based on MIDAS regression. *Economic Modelling*. 50, (2015), 123–129.

[77] Gubanov, M., Stonebraker, M. and Bruckner, D. 2014. Text and structured data fusion in

data tamer at scale. *Data Engineering (ICDE), 2014 IEEE 30th International Conference on* (Mar. 2014), 1258–1261.

[78] Guttman, A. 1984. *R-trees: a dynamic index structure for spatial searching*. ACM.

[79] Haas, L. 2007. Beauty and the Beast : The Theory and Practice of Information Integration. *Database Theory - ICDT 2007: 11th International Conference, Barcelona, Spain, January 10-12, 2007. Proceedings*. T. Schwentick and D. Suciu, eds. Springer-Verlag Berlin Heidelberg. 28 – 43.

[80] Haas, L.M., Hernández, M.A., Ho, H., Popa, L. and Roth, M. 2005. Clio grows up: from research prototype to industrial tool. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*. (2005), 805.

[81] Halevy, a. Y., Ives, Z.G. and Madhavan, J. 2004. The piazza peer data management system. *Knowledge and Data Engineering, IEEE Transactions on*. 16, 07 (Jul. 2004), 787–798.

[82] Halevy, a. Y., Ives, Z.G., Suciu, D. and Tatarinov, I. 2003. Schema mediation in peer data management systems. *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*. (2003), 505–516.

[83] Halevy, A., Franklin, M. and Maier, D. 2006. Principles of dataspace systems. *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '06* (New York, New York, USA, 2006), 1–9.

[84] Halevy, A., Rajaraman, A. and Ordille, J. 2006. Data integration: the teenage years. *Proceedings of the 32nd international conference on Very large data bases* (2006), 9–16.

[85] He, H., Wang, H., Yang, J. and Yu, P.S. 2007. BLINKS: Ranked Keyword Searches on Graphs*. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (2007), 305–316.

[86] Heath, T. and Bizer, C. 2011. Linked Data Evolving the Web into a Global Data Space. *Synthesis lectures on the semantic web: theory and technology*. 1, 1 (Jun. 2011), 1–136.

[87] Hernández, M.A. and Stolfo, S.J. 1995. The merge/purge problem for large databases. *ACM SIGMOD Record* (1995), 127–138.

[88] Hristidis, V. and Papakonstantinou, Y. 2002. Discover: Keyword search in relational databases. *Proceedings of the 28th international conference on Very Large Data Bases* (2002), 670–681.

[89] Hwang, F.K., Richards, D.S. and Winter, P. 1992. *The Steiner tree problem*. Elsevier.

[90] Ihler, E. 1991. Bounds on the quality of approximate solutions to the group Steiner problem. *Graph-theoretic concepts in computer science* (1991), 109–118.

[91] Ikeda, R. and Widom, J. 2009. Data lineage: A survey. *Technical Report. Stanford InfoLab* (2009).

[92] Inmon, W. 2005. *Building the data warehouse*. John wiley & sons.

[93] Ives, Z., Green, T. and Karvounarakis, G. 2008. The ORCHESTRA collaborative data sharing system. *ACM Sigmod Record*. (2008).

[94] Ives, Z., Khandelwal, N., Kapur, A. and Cakir, M. 2005. ORCHESTRA: Rapid, Collaborative Sharing of Dynamic Data. *CIDR* (2005), 107–118.

[95] Jacob, M. and Ives, Z. 2011. Sharing work in keyword search over databases. *Proceedings of the 2011 international conference on Management of data - SIGMOD '11* (New York, New York, USA, 2011), 577–588.

[96] Jensen, C.S. 2008. *Temporal Database Entries for the Springer Encyclopedia of Database Systems*.

[97] Jensen, C.S. 2000. *Temporal database management*. Department of Computer Science,

Aalborg Univerity, Denmark.

[98] Jiang, Y., Li, G., Feng, J. and Li, W. 2014. String Similarity Joins : An Experimental Evaluation. *Vldb*. (2014), 625–636.

[99] Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R. and Karambelkar, H. 2005. Bidirectional expansion for keyword search on graph databases. *VLDB '05 Proceedings of the 31st international conference on Very large data bases* (2005), 505–516.

[100] Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F.M. and Weikum, G. 2009. STAR: Steiner-Tree approximation in relationship graphs. *Proceedings - International Conference on Data Engineering*. (2009), 868–879.

[101] Kaufmann, M., Fischer, P.M., May, N., Ge, C., Goel, A.K. and Kossmann, D. 2015. Bi-temporal Timeline Index: A data structure for Processing Queries on bi-temporal data. *Proceedings - International Conference on Data Engineering*. 2015-May, (2015), 471–482.

[102] Kaufmann, M., Manjili, A. a, Vagenas, P., Fischer, P.M., Kossmann, D., Färber, F. and May, N. 2013. Timeline Index: A Unified Data Structure for Processing Queries on Temporal Data in SAP HANA. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. (2013), 1173–1184.

[103] Kaufmann, M., Vagenas, P., Fischer, P.M., Kossmann, D. and Färber, F. 2013. Comprehensive and interactive temporal query processing with SAP HANA. *Proceedings of the VLDB Endowment*. 6, 12 (2013), 1210–1213.

[104] Kementsietsidis, A. and Miller, J. 2003. Mapping Data in Peer-to-Peer Systems : Semantics and Algorithmic Issues. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. (2003), 325–336.

[105] Khayyat, Z., Lucia, W., Singh, M. and Ouzzani, M. 2016. Lightning Fast and Space Efficient Inequality Joins. *Vldb*. 8, 13 (2016), 2074–2085.

[106] King, G. 2007. An Introduction to the Dataverse Network as an Infrastructure for Data Sharing. *Sociological Methods & Research*. 36, 2 (Nov. 2007), 173–199.

[107] Kirk, T., Levy, A., Sagiv, Y. and Srivastava, D. 1995. The Information Manifold. *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*. (1995), 85–91.

[108] Kllapi, H., Harb, B. and Yu, C. 2014. Near neighbor join. *2014 IEEE 30th International Conference on Data Engineering*. (2014), 1120–1131.

[109] Kou, L., Markowsky, G. and Berman, L. 1981. A fast algorithm for Steiner trees. *Acta informatica*. 15, 2 (1981), 141–145.

[110] Lagoze, C. and Sompel, H. Van De 2001. The Open Archives Initiative: Building a Low-Barrier Interoperability Framework. *First ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'01)*. (2001), 54–62.

[111] Landers, T. and Rosenberg, R.L. 1986. An overview of MULTIBASE. *Distributed systems*. 2, (Jun. 1986), 391–421.

[112] Lee, P.-J.J. 2015. *Efficient information integration system for temporal and spatial data*. University of Pittsburgh.

[113] Lenzerini, M. 2002. Data integration: A theoretical perspective. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2002), 233–246.

[114] Levenstein, V. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*. 1, 1 (1965), 8–17.

[115] Levy, A., Rajaraman, A. and Ordille, J. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. *Technical Report. Stanford InfoLab* (1996).

[116] Li, C., Chang, K.C. and Ilyas, I.F. 2005. RankSQL : Query Algebra and Optimization for Relational Top-k Queries. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (2005), 131–142.

[117] Li, F., Lee, M.L., Hsu, W. and Tan, W.-C. 2015. Linking Temporal Records for Profiling Entities. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*. (2015), 593–605.

[118] Li, F., Yao, B., Tang, M. and Hadjieleftheriou, M. 2013. Spatial Approximate String Search. *IEEE Transactions on Knowledge and Data Engineering*. 25, 6 (Jun. 2013), 1394–1409.

[119] Li, P., Dong, X.L., Maurino, A. and Srivastava, D. 2012. Linking temporal records. *Frontiers of Computer Science in China*. 6, 3 (2012), 293–312.

[120] Linked Data - Design Issues: *http://www.w3.org/DesignIssues/LinkedData.html*. Accessed: 2014-04-09.

[121] Litterman, R.B. 1983. A random walk, Markov model for the distribution of time series. *Journal of Business & Economic Statistics*. 1, 2 (1983), 169–173.

[122] Lo, M.-L. and Ravishankar, C. V 1994. Spatial joins using seeded trees. *ACM SIGMOD Record* (1994), 209–220.

[123] Lorentzos, N.A. and Mitsopoulos, Y.G. 1997. SQL extension for interval data. *IEEE Transactions on Knowledge & Data Engineering*. 3 (1997), 480–499.

[124] Lu, H., Ooi, B.C. and Tan, K.L. 1994. On spatially partitioned temporal join. *Proceedings of the International Conference on Very Large Data Bases*. (1994), 546–546.

[125] Madhavan, J., Jeffery, S., Cohen, S., Dong, X. and Ko, D. 2007. Web-scale data integration:

You can only afford to pay as you go. *CIDR* (2007), 342–350.

[126] Manning, P. 2013. *Big data in history*. Palgrave Macmillan.

[127] McCallum, A., Nigam, K. and Ungar, L.H. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), 169–178.

[128] McCann, R., Shen, W. and Doan, A. 2008. Matching Schemas in Online Communities: A Web 2.0 Approach. *2008 IEEE 24th International Conference on Data Engineering*. (Apr. 2008), 110–119.

[129] Mehlhorn, K. 1988. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*. 27, 3 (1988), 125–128.

[130] Melnik, S., Garcia-Molina, H. and Rahm, E. 2002. Similarity flooding: a versatile graph matching algorithm and itsapplication to schema matching. *Proceedings 18th International Conference on Data Engineering*. (2002).

[131] Michel, S., Triantafillou, P. and Weikum, G. 2005. Minerva∞: A scalable efficient peer-to-peer search engine. *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware* (2005), 60–81.

[132] Microsoft SQL Server Linked Servers: *https://msdn.microsoft.com/en-us/library/ms188279.aspx*. Accessed: 2015-03-28.

[133] Moauro, F. and Savio, G. 2005. Temporal disaggregation using multivariate structural time series models. *The Econometrics Journal*. 8, 2 (2005), 214–234.

[134] Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E. and den Bussche, J. Van 2011. The Open Provenance Model core specification (v1.1). *Future Generation Computer*

Systems. 27, 6 (Jun. 2011), 743–756.

[135] Moreau, L., Miles, S., Missier, P., Simmhan, Y., Futrelle, J., Myers, J., Stephan, E., Kwasnikowska, N., Van den Bussche, J., Freire, J. and Others 2009. The Open Provenance Model (v1. 1). *Future Generation Computer Systems*. 27, (2009), 1–30.

[136] Neo4j: *http://neo4j.com/*. Accessed: 2015-03-28.

[137] Ng, W.S., Ooi, B.C., Tan, K., Peerdb, A.Z. and a 2003. P2P-based System for Distributed Data Sharing. *In Proc. of the 19th International Conference on Data Engineering (ICDE), Bangalore, India, March* (2003), 633–644.

[138] Open Archives, I. 2007. Open Archives Initiative Protocol for Metadata Harvesting. *Open Archives Initiative*. (2007).

[139] Open Archives Initiative: *http://www.openarchives.org/*.

[140] Parameswaran, A., Sarma, A. Das, Garcia-Molina, H., Polyzotis, N. and Widom, J. 2011. Human-Assisted Graph Search : It ' s Okay to Ask Questions. *Proceedings of the VLDB Endowment*. 4, 5 (2011), 267–278.

[141] Parameswaran, A.P. 2011. Answering Queries using Humans , Algorithms and Databases. *Systems Research*. (2011), 160–166.

[142] Paton, N. and Fernandes, A. 2013. Crowdsourcing Feedback for Pay-As-You-Go Data Integration. *DBCrowd 2013*. i, (2013), 1–6.

[143] Pawlik, M. and Augsten, N. 2011. RTED: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment*. 5, 4 (2011), 334–345.

[144] Perez, F. and Granger, B.E. 2007. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*. 9, 3 (2007), 21–29.

[145] Pilourdault, J., Leroy, V., Amer-yahia, S., Pilourdault, J., Leroy, V., Distributed, S.A.,

Pilourdault, J., Leroy, V. and Amer-yahia, S. 2016. Distributed Evaluation of Top-k Temporal Joins. (2016).

[146] Popa, L., Velegrakis, Y., Hernández, M.A., Miller, R.J. and Fagin, R. 2002. Translating web data. *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment.

[147] PrestoDB: *https://prestodb.io/*. Accessed: 2015-03-28.

[148] Qian, L., Cafarella, M.J. and Jagadish, H. V. 2012. Sample-driven schema mapping. *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*. (2012), 73.

[149] Rahm, E. and Bernstein, P. a. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal*. 10, 4 (Dec. 2001), 334–350.

[150] Rahm, E. and Do, H. 2002. COMA: a system for flexible combination of schema matching approaches. *Proceedings of the 28th international conference on Very Large Data Bases* (2002).

[151] Raigoza, J., Sun, J. and Lauderdale, F. 2014. Temporal Join Processing with Hilbert Curve Space Mapping. (2014), 839–844.

[152] Robert J. Hodrick, E.C.P. 1997. Postwar U.S. Business Cycles: An Empirical Investigation. *Journal of Money, Credit and Banking*. 29, 1 (1997), 1–16.

[153] Rossi, N. 1982. A Note on the Estimation of Disaggregate Time Series When The Aggregate is Known. *The Review of Economics and Statistics*. 64, 4 (1982), 695–696.

[154] Roth, M. and Tan, W. 2013. Data Integration and Data Exchange : It ' s Really About Time. *Proceedings of the Sixth Biennial Conference on Innovative Data Systems Research (CIDR 2013)*. (2013).

[155] Ryssevik, J. 2001. The Data Documentation Initiative (DDI) metadata specification. *Ann Arbor, MI: Data Documentation Alliance. Retrieved from http://www. ddialliance. org/sites/default/files/ryssevik_0. pdf.* (2001).

[156] Samet, H. 1990. *The design and analysis of spatial data structures*. Addison-Wesley Reading, MA.

[157] Sarawagi, S. and Bhamidipaty, A. 2002. Interactive deduplication using active learning. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (2002), 269–278.

[158] Sax, C. and Steiner, P. 2013. Temporal Disaggregation of Time Series. *The R Journal*. 5, (2013), 80–87.

[159] Segev, A. 1991. Query processing algorithms for temporal intersection joins. *Data Engineering, 1991. Proceedings. Seventh International Conference on* (1991), 336–344.

[160] Segev, A. and Gunadhi, H. 1989. Event-join optimization in temporal relational databases. *Proc. Int'l. Conf. on Very Large Data Bases* (1989).

[161] Sellis, T., Roussopoulos, N. and Faloutsos, C. 1987. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. (1987).

[162] Shvachko, K., Kuang, H., Radia, S. and Chansler, R. 2010. The Hadoop Distributed File System. *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (May 2010), 1–10.

[163] Silva, J.M.C.S. and Cardoso, F.N. 2001. The Chow-Lin method using dynamic models. *Economic Modelling*. 18, 2 (2001), 269–280.

[164] SLASH2: *http://slash2.psc.edu/*. Accessed: 2015-03-29.

[165] Smith, M., Barton, M., Bass, M., Branschofsky, M., McClellan, G., Stuve, D., Tansley, R.

and Walker, J.H. 2003. DSpace: An open source dynamic digital repository. *D-Lib Magazine*. 9, 1 (2003).

[166] Stonebraker, M., Ilyas, I.F., Zdonik, S., Beskales, G. and Pagan, A. 2013. Data Curation at Scale : The Data Tamer System. *CIDR* (2013).

[167] Stram, D.O. and Wei, W.W.S. 1986. A methodological note on the disaggregation of time series totals. *Journal of Time Series Analysis*. 7, 4 (1986), 293–302.

[168] Strasser, C. 2013. DataUp: Enabling data stewardship for researchers. *IConference* (2013), 657–658.

[169] Systems, D., Marcus, A., Wu, E., Karger, D.R., Madden, S. and Miller, R.C. 2011. Crowdsourced Databases : Query Processing with People Accessed Citable Link Detailed Terms Crowdsourced Databases : Query Processing with People. *CIDR*. (2011).

[170] Talukdar, P. and Jacob, M. 2008. Learning to create data-integrating queries. *Proceedings of the VLDB Endowment*. 1, 1 (2008), 785–796.

[171] Talukdar, P.P., Ives, Z.G. and Pereira, F. 2010. Automatically incorporating new sources in keyword search-based data integration. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (New York, New York, USA, 2010), 387–398.

[172] Talukdar, P.P., Ives, Z.G. and Pereira, F. 2010. Automatically incorporating new sources in keyword search-based data integration. *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*. (2010), 387.

[173] Tamr: *http://www.tamr.com/*.

[174] Tang, M., Yu, Y., Aref, W.G., Malluhi, Q.M. and Ouzzani, M. 2015. Efficient Processing of Hamming-Distance-Based Similarity-Search Queries Over MapReduce. *Edbt*. (2015),

361–372.

[175] Taylor, N.E. and Ives, Z.G. 2006. Reconciling while tolerating disagreement in collaborative data sharing. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (New York, New York, USA, 2006), 13–24.

[176] Tejada, S., Knoblock, C.A. and Minton, S. 2002. Learning domain-independent string transformation weights for high accuracy object identification. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (2002), 350–359.

[177] Temporal Disaggregation Library: 2013. *http://www.mathworks.com/matlabcentral/fileexchange/39770-temporal-disaggregation-library*.

[178] Theobald, M., Schenkel, R. and Weikum, G. 2005. An efficient and versatile query engine for TopX search. *VLDB 05 Proceedings of the 31st international conference on Very large data bases* (2005), 625–636.

[179] Toman, D. 1998. Point-Based Temporal Extensions of SQL and Their Efficient Implementation. *International Conference on Deductive and Object-Oriented Databases*. (1998), 211–237.

[180] Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E. and de Walle, R. 2014. Web-scale querying through linked data fragments. *Proceedings of the 7th Workshop on Linked Data on the Web* (2014).

[181] Verborgh, R. and Wilde, M. De 2013. *Using OpenRefine*. Packt Publishing Ltd.

[182] Vlachou, A., Doulkeridis, C., Nørvåg, K. and Kotidis, Y. 2013. Branch-and-bound algorithm for reverse top-k queries. *Proceedings of the 2013 international conference on*

Management of data (New York, New York, USA, 2013), 481–492.

[183] Wang, J., Kraska, T., Franklin, M.J. and Feng, J. 2012. CrowdER: crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*. 5, 11 (Jul. 2012), 1483–1494.

[184] Wang, J., Li, G. and Feng, J. 2012. Can we beat the prefix filtering? An Adaptive Framework for Similarity Join and Search. *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*. (2012), 85.

[185] Weibel, S., Kunze, J., Lagoze, C. and Wolf, M. 1998. Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*. 2413, 222 (1998), 132.

[186] Wijsen, J. 2009. Consistent query answering under primary keys: a characterization of tractable queries. *Proceedings of the 12th International Conference on Database Theory* (2009), 42–52.

[187] Winkler, W.E. 2006. Overview of record linkage and current research directions. *Bureau of the Census* (2006).

[188] Winter, P. and Smith, J.M. 1992. Path-distance heuristics for the Steiner problem in undirected networks. *Algorithmica*. 7, 1-6 (1992), 309–327.

[189] Yen, J.Y. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science*. 17, 11 (1971), 712–716.

[190] Zadorozhny, V. and Hsu, Y.-F. 2011. Conflict-aware historical data fusion. *Scalable Uncertainty Management(SUM)*. (2011), 331–345.

[191] Zadorozhny, V., Manning, P., Bain, D.J. and Mostern, R. 2013. Collaborative for Historical Information and Analysis: Vision and Work Plan. *Journal of World-Historical Information*. 1, 1 (2013), 1–14.

[192] Zadorozhny, V. and Raschid, L. 2002. Query optimization to meet performance targets for

wide area applications. *Proceedings 22nd International Conference on Distributed Computing Systems*. (2002).

[193] Zadorozhny, V., Raschid, L. and Gal, A. 2008. Scalable Catalog Infrastructure for Managing Access Costs and Source Selection in Wide Area Networks. *International Journal of Cooperative Information Systems*. 17, 01 (2008), 77–109.

[194] Zadorozhny, V., Raschid, L., Vidal, M.E., Urhan, T. and Bright, L. 2002. Efficient evaluation of queries in a mediator for WebSources. *ACM SIGMOD international conference on Management of data* (2002), 85–96.

[195] Zaier, L.H. and Trabelsi, A. 2007. A Polynomial Method for Temporal Disaggregation of Multivariate Time Series. *Communications in Statistics - Simulation and Computation*. 36, 3 (2007), 741–759.

[196] Zhang, D.Z.D., Tsotras, V.J. and Seeger, B. 2002. Efficient temporal join processing using indices. *Proceedings 18th International Conference on Data Engineering*. (2002).

[197] Zhong, M., Moore, J., Shen, K. and Murphy, A.L. 2005. An Evaluation and Comparison of Current Peer-to-Peer Full-Text Keyword Search Techniques. *WebDB* (2005), 61–66.

[198] Zhu, H., Madnick, S.E. and Siegel, M.D. 2004. Effective data integration in the presence of temporal semantic conflicts. *Temporal Representation and Reasoning, 2004. TIME 2004. Proceedings. 11th International Symposium on* (2004), 109–114.

[199] Zurek, T. 1997. Optimisation of partitioned temporal joins. *Advances in Databases*. (1997), 101–115.