DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Generating Labeled Datasets For Schema Matching

Author:
Konstantinos CHRONAS

Supervisor:
Dr. Asterios KATSIFODIMOS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

Student number: 4923162

Thesis committee:

Dr. A. Katsifodimos,	TU Delft, chair
Dr. H. Wang,	TU Delft
Dr. J. Yang,	TU Delft

An electronic version of this thesis is available at
https://repository.tudelft.nl/.

March 19, 2023

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Software Technology

Master of Science

Generating Labeled Datasets For Schema Matching

by Konstantinos CHRONAS

Matching schemas is a fundamental task in data integration and semantic web applications. However, generating labeled data for schema matching tasks is challenging, requiring , requiring an efficient and effective approach. This thesis addresses this challenge by investigating schema matching techniques and crowdsourcing solutions. We developed a prototype crowdsourcing platform for schema matching called Crowdie. The platform utilizes a novel pre-filtering algorithm to reduce the number of possible correspondences and improve the platform's efficiency while minimizing the cost of crowdsourcing. Additionally, we designed a simple yet effective task interface to ensure high-quality labeled data. Our findings demonstrate that crowdsourcing is viable for generating labeled data for schema matching tasks. Overall, this work contributes to reducing search spaces and developing crowdsourcing solutions for schema matching tasks.

Acknowledgements

I express my deepest gratitude to my supervisor Asterios for giving me the opportunity to work on this project. His guidance and expertise were invaluable in shaping this research work.

Additionally, I would like to thank my friends for their unwavering support throughout the project, as their encouragement kept me motivated to push through the challenges.

Finally, I would like to extend my heartfelt appreciation to my family for their unending love and support, which has been a constant source of strength and inspiration in helping me achieve my goals.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1
- 1.1 Problem Statement . . . 2
  - 1.1.1 Efficiency in Schema Matching . . . 2
  - 1.1.2 Effectiveness in Schema Matching . . . 3
- 1.2 Research Questions . . . 4
- 1.3 Contributions . . . 5
- 1.4 Outline . . . 5

2 Literature Review 7
- 2.1 Schema Matching Techniques . . . 8
  - 2.1.1 Schema-based techniques . . . 8
- 2.2 Schema matching systems . . . 9
  - 2.2.1 Benchmarks . . . 10
- 2.3 Crowdsourcing . . . 11
  - 2.3.1 Crowdsourcing in Schema matching . . . 12

3 Dimensionality Reduction 15
- 3.1 Pre-Filtering Technique . . . 16
  - 3.1.1 Contraint-based technique . . . 16
  - 3.1.2 Text Processing . . . 17
  - 3.1.3 Linguistic Technique and External resource . . . 17
- 3.2 Pre-filtering Technique In Action . . . 19
  - 3.2.1 Datasets . . . 20
  - 3.2.2 Evaluation . . . 20
- 3.3 Experiments . . . 22
  - 3.3.1 Constraint-based technique experiments . . . 24

4 Crowdie: a crowdsourcing platform for schema matching 27
- 4.1 System Architecture . . . 28
  - 4.1.1 Pre-Filtering Module . . . 28
  - 4.1.2 Task Creation Module . . . 29
  - 4.1.3 Crowdie's User Interface . . . 29
  - 4.1.4 Aggregation Module . . . 32
- 4.2 Task Design . . . 32
- 4.3 Quality Control . . . 34
- 4.4 Evaluation . . . 35
- 4.5 Results . . . 36

5 Conclusion **39**
5.1 Summary 39
5.2 Discussion Future Directions 40
5.2.1 Search Space Reduction in Schema Matching 41
5.2.2 Crowdie 41

A Appendix A **43**
A.1 Part of the results for grid search the new york and la data 43

List of Figures

1.1 Pairwise schema matching Rahm and Peukert [2019] 2

2.1 Taxonomy of schema matching techniquesRahm and Bernstein [2001] . 8
2.2 Taxonomy of schema matching techniques Shvaiko and Euzenat [2005] 9
2.3 CrowdMacther system architecture Zhang et al. [2014] 12
2.4 Question designs for schema matching Hung et al. [2013] 13

3.1 Number of missed correct correspondences for each method. 22
3.2 The New York Data threshold distribution of the two methods 24
3.3 The Los Angeles Data threshold distribution of the two methods . . . 24

4.1 Crowdie: System Architecture. 28
4.2 Crowdie: Welcome Page 30
4.3 Crowdie: Sign In 30
4.4 Crowdie: About Page 31
4.5 Example microtask: Workers are asked to match columns between two tables. 31
4.6 Tasks Page: Available Tasks and Load Microtasks 32
4.7 Tasks Page: Success and Continue 32
4.8 Task Design of CrowdMatcher Zhang et al. [2014] 33
4.9 Example microtask: Workers are asked to match columns between two tables. 33
4.10 Example microtask: Workers are asked to match columns between two tables. 34

A.1 Threshold Exploration Cosine-w2v NY Data 43
A.2 Threshold Exploration Cosine-w2v LA Data 44
A.3 Threshold Exploration Jaro-Winkler NY Data 45
A.4 Threshold Exploration Jaro-Winkler LA Data 46
A.5 Threshold Exploration Levenshtein NY Data 47
A.6 Threshold Exploration Levenshtein LA Data 48

List of Tables

3.1 Number of tables and the labeled combinations for each domain 20
3.2 Wikidata Threshold Exploration Results 23
3.3 Average results of the wikidatas' table combinations 23
3.4 Average results of the New Table table combinations 23
3.5 Average results of the Los Angeles table pairs 23
3.6 Results of the table combinations that required the minimum threshold for each case. 1) refers to Cosine-w2v, and 2)Cosine-w2v with the constraint reduction technique 25
3.7 asdasd 25

4.1 Number of people who participated in the pilot study with their knowledge background in computer science. 35
4.2 The tasks that were used in the pilot study. 35
4.3 Number of answers each worker provided for their tasks with the number for labels of that particular task. 36
4.4 Shows the column names of the labeled microtasks from the workers and whether they are correctly labeled 37

Chapter 1

Introduction

In today's business landscape, companies and organizations are experiencing an unprecedented surge in the amount and diversity of data they need to handle to meet their business requirements Nargesian et al. [2019]. Additionally, the variety of data has broadened, arising from various end-user devices and systems or different teams within the same company that adopts a more flexible, microservices-oriented design, resulting in other formats.

Traditionally, organizations used data warehouses as a central repository for storing data to meet their business needs. However, they can be a bottleneck in large-scale machine learning and predictive analytics applications due to the high velocity and volume of data, which makes the curation process cumbersome and impractical. In such cases, data lakes have emerged as the preferred option. The primary difference between the two is that data is stored directly in the system, leaving the curation process to the end user Koutras et al. [2021].

Nowadays, almost all complex data science tasks require data integration. The primary aspect of data integration is dataset discovery, which involves searching through multiple data sources to locate relevant datasets and determine how they relate to one another. Data discovery is a critical component of an ML pipeline. It helps ensure that the data used to train and test machine learning models is accurate, relevant, and representative of real-world scenarios. In Nargesian et al. [2019], the authors listed data discovery as one of the biggest challenges in data lake management, highlighting the need and challenge of on-demand schema matching and mapping.

An essential part of data discovery is schema matching. When dealing with tabular data, finding relevant datasets involves using schema matching methods to automatically determine whether columns or entire tables can be combined or merged. These methods rely on information about how datasets are related to each other, and the accuracy of the matching technique can significantly impact the effectiveness of the data discovery process. The exponential data volume and diversity growth increased the demand for schema matching.

However, this process has its challenges, and the lack of a labeled dataset is one of the most significant obstacles to accurately matching schemas. Unfortunately, the absence of a labeled dataset is a common problem in many real-world data integration scenarios, making it difficult to achieve the necessary accuracy in schema matching. Thus the primary purpose of this work is to propose a pipeline for acquiring labeled datasets for schema matching. Labeled datasets can improve schema matching systems' effectiveness.

1.1 Problem Statement

Schema matching aims to identify semantic correspondences between metadata structures or models (schemas, XML, ontologies) Rahm and Bernstein [2001]. Achieving this goal can be a complex and challenging task. The schemas can be acquired from different data sources with differences in structures, data types, and semantics. Additionally, there can be multiple possible correspondences between elements in different schemas, which makes it difficult to determine the correct one. Hence this semantic heterogeneity and ambiguity in different schemas make it difficult for automated approaches to identify the correspondence correctly. The computational cost of these algorithms also increases complexity when dealing with large schemas with thousands of elements or many data sources with different schemas.

Schema matching can be performed using pairwise or holistic matching Rahm and Peukert [2019]. Holistic schema matching is a technique for matching the overall structure or schema of two or more data sources to determine equivalent structures and facilitate integration and interoperability. Unlike holistic schema matching, which compares the overall structure of multiple schemas, pairwise schema matching focuses on comparing two schemas at a time. The goal of pairwise schema matching is to identify equivalent or similar elements and relationships between elements in the two schemas. The result of pairwise schema matching is usually an equivalence mapping containing the identified semantic correspondences. Algorithms can determine approximate mappings only due to the semantic heterogeneity of schemas. Therefore, obtaining the correct mapping requires the knowledge of human domain experts. Furthermore, the results are helpful for data integration tasks such as merging or integrating the respective schemas. Additionally, most recent research has shifted from generic schema matching to the more specific domain, such as joinable Zhu et al. [2019] or unionable Nargesian et al. [2018] attribute search, proving that the scope of the problem is essential in the method design.

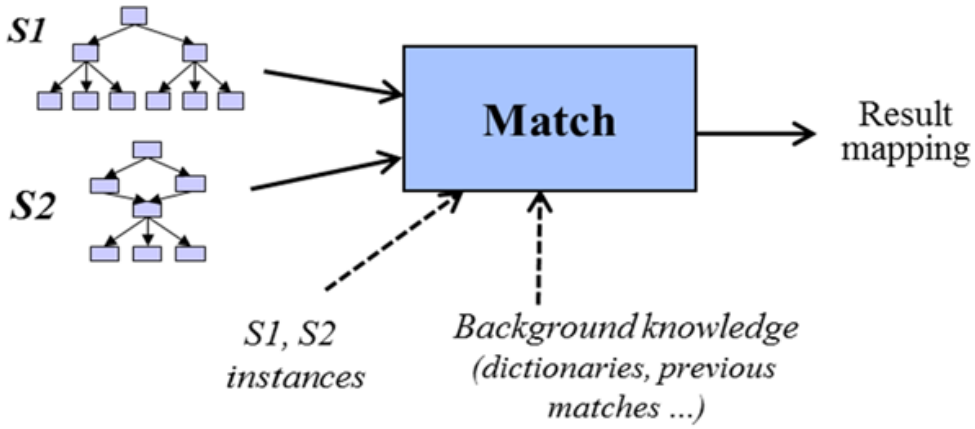

FIGURE 1.1: Pairwise schema matching Rahm and Peukert [2019]

According to Rahm and Peukert, automated systems still struggle to achieve good efficiency and effectiveness despite the evolution of schema matching systems.

1.1.1 Efficiency in Schema Matching

Efficiency refers to the main performance problem for matching large schemas is the potentially vast search space. Comparing every element of the input schema with

the source may lead to a high computational cost, especially when the two schemata contain thousands of elements. In math terms, this could lead to O^n number of pairs where O refers to the number of elements of the input schema and n to the number of elements of the target. Therefore, improving the efficiency of schema matching systems for large-scale matching is needed by reducing the search space or employing parallel computation to reduce the runtime or a combination of both.

Parallel computation is used to speed up single matchers by partitioning the search space and evaluating different partitions in parallel on different processors Koutras et al. [2021], Amin et al. [2016], Gross et al. [2010]. Reducing the search space has two potential approaches. The first is applying several schema matching techniques requiring a low runtime speed and a low similarity threshold. This can eliminate element pairs below the threshold before applying other matching techniques. The second one is to partition the input schemas into subsets and restrict similar schema elements to a subset of these partitions Aumueller et al. [2005].

Improving the efficiency of schema matching systems still holds a significant challenge, especially in pruning distinct elements. Previous pruning techniques have been used in Ehrig and Staab [2004] and Peukert et al. [2010]. However, since 2010 there have been no significant updates to the subject. Thus the first part of this work will focus on creating a new low-cost matching technique to reduce the dimensionality of schema matching pairs which will then be compared to previously used matching techniques.

1.1.2 Effectiveness in Schema Matching

Schema-matching systems need help finding the correct and complete identification of semantic correspondences. This problem becomes more difficult for larger search spaces as the ambiguity and heterogeneity usually increase the possibility of identifying false match candidates also increases. Researchers over the years have been trying to improve the effectiveness by utilizing background knowledge from domain-specific thesauri and dictionaries. Schema matching tools like Artemis Castano and De Antonellis [2001] and Cupid Madhavan et al. [2001] have used thesauri and dictionaries to produce more effective results. Another way is to reuse the previous match results from a maintained repository to utilize background knowledge. This method has been used by Comma Aumueller et al. [2005] and Gomma Gross et al. [2010]. Although it holds promise, it requires a complex infrastructure with persistent storage requirements to infer previous correspondences. Some systems employ a semi-automatic tunic of match strategies. These approaches use learning-based or rule-based models. However, this approach requires many labeled data to discover the optimal parameter settings.

Schema matching has been viewed by Bernstein et al. as an AI-complete problem. Automated systems, for example, can not make a distinction between country names and country codes ("Netherlands", "NL"), which guide to the same thing. Schema matching has been viewed by Bernstein et al. as an AI-complete problem. For example, automated systems can not make a distinction between country names("Netherlands") and country codes ("NL"), which guide to the same thing. Thus, post-processing and repair of the suggested correspondences by domain experts are required to improve the effectiveness. In general, schema matching tools utilize a user interface displaying the match results back to the user, indicating the similarities between the possible correspondences with different colors, such as red and green. However, this approach has failed when dealing with large schemata where the number of possible matching pairs radically increases. Attempts have

been made to include the verification of schema matching results as part of crowdsourcing in Zhang et al. [2014], Hung et al. [2013]. The possible correspondences are formed as microtasks for the crowd workers to answer.

While crowdsourcing could be a good solution for distributing the verification workload to multiple workers, it comes with other problems, such as preserving the quality of the results, keeping the cost low, and making task design choices. Therefore, the second part of this work will focus on creating a user interface for crowdsourcing schema matching tasks to improve the verification process and generate labeled datasets that could be used to benchmark the schema matching systems to make them more robust.

1.2 Research Questions

The problem statement states that the schema matching process struggles with efficiency and effectiveness (see section 1.1). Thus, acquiring labeled datasets from multiple domains is essential for enhancing the efficiency and effectiveness of automated schema matching tools. In previous years, researchers have been manually labeling datasets to test the performance of their systems. However, this process is time-consuming due to the problems of schema matching. Therefore, we derived our main research question:

> How to create a system architecture for a crowdsourcing platform that generates labeled datasets for schema matching systems?

To answer this question, a system for acquiring labeled datasets has to consider the efficiency and effectiveness of schema matching. First, we must address the efficiency problem derived from the potentially large search space. The focus is to reduce the search space by using a cost-efficient technique. Reducing the search space has been done previously in Peukert et al. [2010]. However, not a lot of work has been done since then, leading to our first sub-research question:

> How to create a pre-filtering schema matching technique for search space reduction?

A search space reduction technique aims to reduce the number of possible matchings without excluding actual matchings. Researchers used a low similarity threshold based on their intuition to achieve that. Thus, we need to identify the best threshold for our technique by performing a grid search using previously labeled data. Additionally, we need to compare the performance of our technique with other techniques used for search space reduction by creating a benchmark test. These lead to our second sub-research question:

> What is the optimal threshold value for search space reduction techniques?

Schema matching systems previously required human verification to improve their results' effectiveness. Some works addressed the problem of verifying schema matching results as a crowdsourcing problem . For our case, we will also use crowdsourcing as the second part of our platform to acquire labeled data. Crowdmatcher Zhang et al. [2014] is one prominent example of using crowdsourcing in schema

matching. However, the task design does not contain contextual information, which leads to our third sub-research question:

> What is considered a task, and how to design a task for schema matching?

Finally, we need to create an aggregation algorithm to eliminate wrong answers, which leads to our final sub-research question:

> How to aggregate the results from the crowd?

1.3 Contributions

This section outlines the contributions that emerged while answering our research questions in the same order, as expressed in section 1.2.

1. A novel pre-filtering algorithm for reducing the number of possible matching pairs given a threshold. This algorithm utilizes a combination of schema matching techniques that operate on the element level, thereby improving the efficiency of the schema matching process.

2. A value suggestion for the threshold for removing attribute pairs, which can control the recall of the filtering process to ensure that no actual matches are removed. This approach can refine the matching results by filtering out irrelevant pairs while preserving potentially valuable pairs for further evaluation.

3. A novel crowdsourcing task design for annotating similar columns for schema matching.

4. A crowdsourcing prototype platform for schema matching tasks that can generate labeled data and verify the results of schema matching systems. This platform allows users to annotate schema elements, generating high-quality labeled data for evaluating schema matching systems. Additionally, the platform can be used to verify the results of schema matching systems, improving the overall effectiveness of the matching process.

1.4 Outline

The rest of the thesis is structured as follows. In chapter 2, we conduct a literature review on schema matching to discover schema-matching techniques for search space reduction. Additionally, we list a couple of schema matching systems to summarize how they work. Finally, we check how crowdsourcing is utilized for schema matching.

The chapter 3 details the design of our search space reduction technique. In addition, the chapter explains how we conducted the experiments and which methods we used for our comparison. Finally, it also presents the results of the experiments.

The chapter 4 showcases the crowdsourcing platform. It details the design of the platform, the task design choices, and the aggregation algorithm for the crowd answers. Additionally, the chapter presents the experiments that we used.

We conclude this work in chapter 5 by summarizing our work, its limitations, and showcasing future directions

Chapter 2

Literature Review

In this chapter, we will explore the existing research on schema matching, which is a crucial step in data integration and interoperability. To answer our first sub-research question, we will start by examining the various schema-matching techniques that have been used over the years in section 2.1. We will review these techniques to identify those that could be used to address the research question at hand. Next, we will delve into the architecture of schema systems in section 2.2. This section will provide an overview of the different types of schema systems and their components. We will also look at some benchmark systems for schema matching to understand the state-of-the-art in this field. Finally, we will review the literature on crowdsourcing for schema matching in section 2.3. This section will highlight the various approaches for leveraging human intelligence for schema matching tasks. We will also examine the benefits and limitations of crowdsourcing for schema matching so that we will be able to create our crowdsourcing platform for generating labeled datasets to answer our main research question.

2.1 Schema Matching Techniques

Over the years, the database community has extensively researched various schema-matching techniques and combinations. The initial automated techniques were reviewed in Rahm and Bernstein [2001], where the authors presented a taxonomy that categorizes the approaches based on using a single matching technique or a combination of different ones. They also differentiated schema-level and instance-level matching techniques. Schema-level techniques operate on schema information, while instance-level techniques associate with the data content. The taxonomy is detailed in figure 2.1.

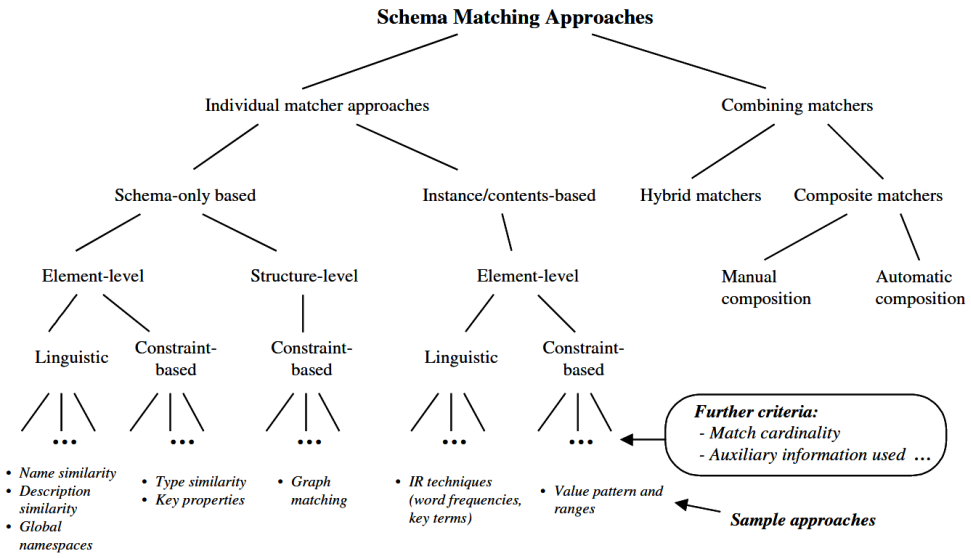

FIGURE 2.1: Taxonomy of schema matching techniquesRahm and Bernstein [2001]

Another review introduced in Shvaiko and Euzenat [2005] expanded the taxonomy presented in Rahm and Bernstein [2001] to incorporate into the input information three new classes the syntactic, the semantic, and the external class. These classes were then separated into terminological, structural, and semantic approaches (see fig: 2.2)

In recent years schema matching surveys have somewhat agreed with the taxonomy presented in Shvaiko and Euzenat [2005]. Rahm and Bernstein presented an updated survey of the developments in schema-matching techniques Bernstein et al. [2011]. They also included techniques for user interaction and feedback in the matching process. In Alwan et al. [2017], the authors showcased a detailed classification of schema matching approaches designed for instance-based schema matching.

2.1.1 Schema-based techniques

This work will focus on schema-based matching techniques linked to element-level matching. According to Bernstein et al. [2011], Element level matching seeks to match the elements belonging to the source schema with elements of the input target schema. In many cases, it is possible to exploit the schema elements at the finest level, such as attributes in an XML schema or attributes in a relational schema.

- **Linguistic-based techniques** affect the linguistic information of the database schemas, such as the attribute names to determine the match between the input

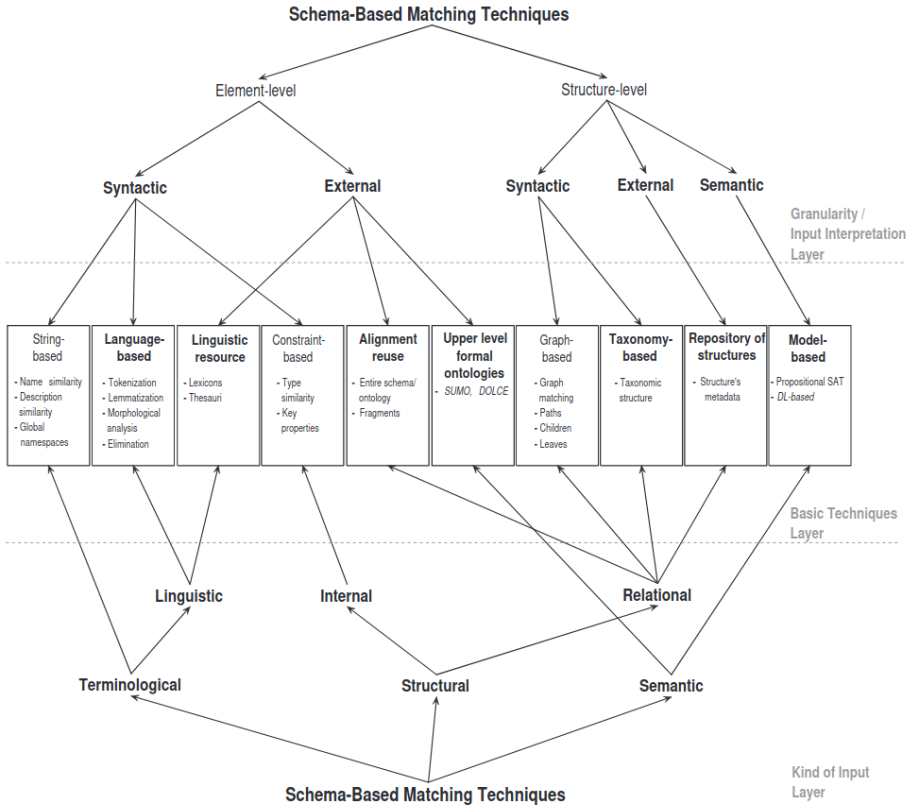

FIGURE 2.2: Taxonomy of schema matching techniques Shvaiko and Euzenat [2005]

and the source schema. The match is accomplished using stemming, tokenization, string and substring matching, and information retrieval techniques.

- **External linguistic resources** are used to explore the linguistic relations between the words (for example, synonyms and hyponyms). This is achieved using common knowledge or domain-specific thesauri, dictionaries, and mismatch lists.

- **Constraint-based techniques** are algorithms that deal with the internal information of a database, such as the data types, the range of values, the uniqueness, and the cardinality of attributes and keys.

2.2 Schema matching systems

Schema matching has been challenging over the years, and researchers have created different systems to automate this task Bernstein et al. [2011]. This work will focus on systems that work with schemata of any kind (i.e., relational, XML) without the requirement for an external ontology and use some String, language, or linguistic techniques for matching Shvaiko and Euzenat [2005]. Some notable schema matching systems are the following:

Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) Castano and De Antonellis [2001] is a schema-matching system that serves affinity-based analysis and hierarchical clustering of source schema elements. The affinity-based analysis represents the matching step: in a hybrid manner, it calculates

the name, structural, and global affinity coefficients exploiting a common dictionary. Using the global affinity coefficients, a hierarchical clustering technique categorizes classes into groups at different levels of affinity. Each cluster creates a set of global attributes - a global class. Logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table.

Similarity Flooding (SF) Melnik et al. [2002] is a schema matching system that can be used with relational, RDF, and XML schemas. The input data are transformed into labeled graphs, and the system manipulates them using a fix-point computation to create an alignment between the then nodes of the graph. The system uses a string-based comparison of the vertices labels to obtain the initial correspondences between the graph nodes, which is also refined within the fix-point computation. The technique starts with a string-based comparison (common prefix, suffix tests) of the vertices labels to obtain an initial alignment which is refined within the fix-point computation. The algorithm uses propagation coefficients to spread similarity between similar nodes and adjacent neighbors. This process runs until the similarity measure is increased till the fix-point is reached.

Cupid Madhavan et al. [2001] is a hybrid system consisting of linguistic and structural schema matching techniques and computes the similarity coefficients with the help of a domain-specific dictionary. It is designed to be generic across data models and has been applied to XML and relational examples. Cupid has three phases. The first phase does linguistic element-level matching and categorizes elements based on names, data types, and domains. The second phase transforms the original schema into a tree and then does a bottom-up structure matching, resulting in a structural similarity between pairs of elements. The third phase calculates a weighted mean based on the linguistic and structural similarity of the pairs, and then this weighted mean is used to decide on a mapping higher than the given threshold.

COMA++(Combination Of Matching Algorithms) Aumueller et al. [2005] is the extended version of the previous prototype COMA Do and Rahm [2002]. COMA is a hybrid matching system that considers many linguistic and structural matching techniques, including various forms of reusing previously determined match results. A fragment-based match approach also disintegrates a large match problem into smaller problems. This system can process Relational, XML, and RDF schemas and ontologies. Coma transforms the input schema into a rooted, directed acrylic graph. This transformation aims at capturing contexts in which the elements occur. The final step involves combining the similarity values given by each similarity measure provided by the user using aggregation operators. Finally, COMA++ shows all the possible matching pairs whose similarity value is above a given threshold, and the user checks and validates their accuracy.

Schema matching systems also employ machine learning techniques to automate the matching process based on previously known mappings. Notable examples are LSD Doan et al. [2001], and corpus-based schema matching Madhavan et al. [2005].

2.2.1 Benchmarks

The first attempt at a schema matching benchmark is XBenchMatch Duchateau et al. [2007], which introduced a preliminary system for evaluating schema matching tools. Nevertheless, it showcased three schema-based methods focusing only on XML data. XBenchMatch received an update Duchateau and Bellahsene [2014] with more XML datasets, additional methods, and performance metrics, making XBenchMatch a

more thorough benchmark. However, it must be completed as it excludes instance-based methods and other data formats. XML is standard in many websites and applications but is less popular in the database domain.

Another attempt for benchmarking schema matching systems is Valentine Koutras et al. [2021]. Valentine is an open-source experiment suite that facilitates large-scale automated matching experiments on tabular data. It incorporates implementations of key schema matching methods developed from scratch or imported from open-source repositories. Additionally, it provides a robust framework for executing and organizing schema matching experiments, allowing researchers to evaluate and systematically compare different techniques quickly.

2.3 Crowdsourcing

As mentioned above, schema matching holds effectiveness problems for data integration. Automated tools cannot produce accurate results without facing uncertainty problems. These problems arise from the schema's inability to fully include the represented data's semantics. A solution to reduce the uncertainty is to leverage crowdsourcing platforms and ask humans to correct any mistakes made by the automated toolsZhang et al. [2013].

Crowdsourcing has been a common practice for computer-hard tasks in many subjects, such as sentiment analysis Borromeo and Toyama [2015], image recognitionArganda-Carreras et al. [2015], and entity resolutionWang et al. [2012]. It can also benefit data management applications, such as data cleaning Wang et al. [2012], data integration Demartini et al. [2013], and knowledge construction Amsterdamer et al. [2015]. Public crowdsourcing platforms such as Amazon's Mechanical Turk(AMT) [1] and Crowdflower [2] made crowdsourcing easily accessible. These platforms have two types of users requesters and workers. Requesters are the ones who publish the tasks on a crowdsourcing platform, and the workers are the ones who perform the tasks. In detail, requesters first need to design the tasks. This process is called task design. Task design usually refers to the design of the user interface for the task, determining the type and requirements of the task Zheng et al. [2011]. The task types can vary between single-choice, multiple-choice, fill-in-the-blank, and collection. The requirements refer to the task settings, including the price and time required to complete the tasks Haas et al. [2015], Zhang et al. [2013]. Additionally, the requester needs to set quality control techniques Daniel et al. [2018], which can either be selected from the platform or created using their methods. After the tasks are published to the platform by the requester, workers can then answer them and submit the answers to the selected platform Li et al. [2017].

Three core crowdsourced data management techniques must be considered: quality control, cost control, and latency control. Quality control operates to obtain high-quality answers from the crowd workers, by categorizing a worker's quality and aggregating their answers Daniel et al. [2018]. Cost control aims to reduce human costs while still keeping good result quality. Latency control exploits how to reduce the latency by modeling. There is always a trade-off between these three techniques. Thus, research studies often focus on the balance between them Li et al. [2017].

[1] https://www.mturk.com/

[2] http://www.crowdflower.com

2.3.1 Crowdsourcing in Schema matching

Crowdsourcing for schema matching can potentially improve the accuracy and coverage of schema matching, especially when dealing with large numbers of schemas or with highly heterogeneous schemas that change frequently. However, one of the main challenges of using crowdsourcing for schema matching is ensuring the annotations' quality and managing the cost of the crowdsourcing process.

Additionally, it's also important to remember that crowdsourcing can be a cost-effective way of obtaining labeled data and resolving ambiguities. Schema matching algorithms usually produce many possible matches that require labeling between two given schemata for a single worker to label all of them. Therefore, in Zhang et al. [2013] the authors suggested two frameworks to reduce the uncertainty in schema-matching tools using crowdsourcing. The tasks include a sequence of Correspondence Correctness Queries (CCQ), which require a simple binary answer from the crowd to determine the relevancy of the correspondence. The Single CCQ (Correspondence Correctness Question) and Multiple CCQ. Both frameworks take advantage of the Shannon entropy measure to reduce the uncertainty since the result pairs are usually associated with the probability of being an actual match. Single CCQ publishes one query with the highest entropy at a time to the crowd workers to get a binary yes or no answer. In addition, Multiple CCQ is an extension of Single CCQ, where the crowd workers must answer the top k most contributing queries based on the entropy measure in parallel.

Zhang et al. incorporated their work in Zhang et al. [2013] in a hybrid machine-crowd system called CrowdMatcherZhang et al. [2014]. This system incorporates schema and attributes clustering techniques when the user provides multiple schemata to create a mediated schema. It also includes automated schema matching tools and the frameworks mentioned in Zhang et al. [2013] to minimize the cost of crowdsourcing and reduce uncertainty. Figure 2.3 shows an overview of the system architecture.

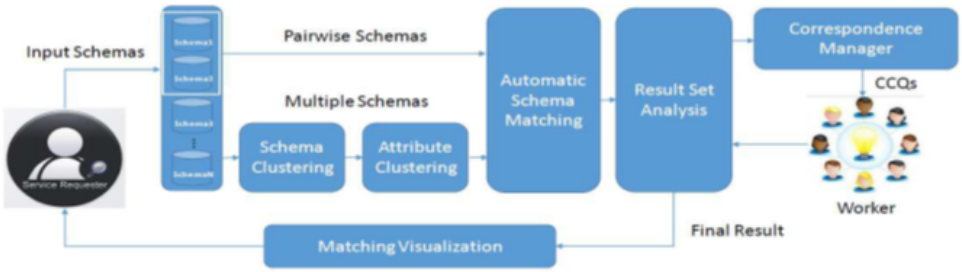

FIGURE 2.3: CrowdMacther system architecture Zhang et al. [2014]

The correspondences generated by automated schema-matching tools are used as queries for the crowd. Although, a simple task design of such a question could be: Does table1.column1 contain the same meaning as table2.column1? The authors in Hung et al. [2013] suggested that providing contextual information about the matching problem to the questions can guide the crowd workers and increase the accuracy of their answers. An example of the task design can be seen in figure 2.4. Additionally, they used several constraints to adjust the error rate and reduce worker efforts.

Fan et al. in Fan et al. [2014] explored a machine-crowdsourcing framework for the web table schema matching problem. They designed a graph-based approach to

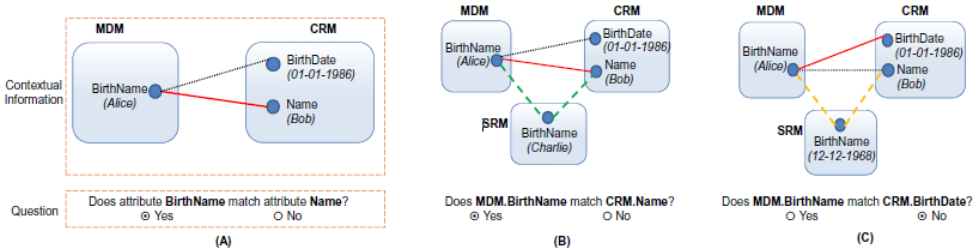

FIGURE 2.4: Question designs for schema matching Hung et al. [2013]

capture the columns related to the same concept. Then they proposed an expected utility algorithm that selects the top k columns that are more beneficial for the crowd to annotate, which reduces the cost of crowdsourcing. However, their work did not focus on the quality of the given answers to satisfy the required accuracy for the minimal number of workers.

Chapter 3

Dimensionality Reduction

This chapter presents a novel pre-filtering schema matching technique to reduce the search space for finding correct correspondences in tabular data. To answer the first sub-research question on creating a pre-filtering schema matching technique for search space reduction, we detail the methods used for the pre-filtering algorithm in Section 3.1. The selected methods, datasets, and evaluation metrics used to benchmark the proposed method are presented in Section 3.2. Finally, the chapter concludes in Section 3.3 by showcasing the results from the evaluation, which demonstrate the effectiveness of the proposed approach and provides an answer to our second sub-research question: "What is the optimal threshold value for search space reduction techniques?".

3.1 Pre-Filtering Technique

This new schema matching technique aims to reduce the number of possible pairs that a schema matching system uses as input to find the correct correspondences. The design of such a technique has to take into account two considerations. First, the schema matching methods used previously for this purpose Peukert et al. [2010], and the category in which they belong according to the literature Bernstein et al. [2011]. Second, the complexity of the technique. Schema matching is already a computationally expensive process, and thus, the technique should avoid adding more complexity to the process.

Techniques with low runtime speed have to be selected. Schema-based approaches that operate on the element level are usually cheap to use. According to the taxonomy in Bernstein et al. [2011], these approaches belong to the linguistic, external resources, and constraint classes. The goal of our suggested technique is to reduce the number of pairs without losing possible correspondences between two different schemata using a similarity measure with a low threshold. To achieve that, we propose combining schema matching techniques for the classes mentioned above, which are used sequentially.

First, the algorithm finds all the possible attribute pairs between two tables. Then, it applies a constraint-based technique to filter out the combination of attribute pairs having dissimilar data types. Natural Language Processing approaches are used to process the text of the chosen attribute names. Additionally, an external resource is utilized to discover the semantic relations between the processed text of the attribute names. Finally, a similarity measure is employed to calculate the similarity between the selected pairs and remove the ones that reside below the given threshold.

3.1.1 Contraint-based technique

The first technique of our method utilizes a constraint-based approach. Constraint-based approaches are algorithms that deal with the internal information of a database, such as the data types, the range of values, the uniqueness, and the cardinality of attributes and keys Bernstein et al. [2011]. This work will explore only the information regarding the data types of the columns in a tabular dataset. Tabular data types can be the following:

- Numeric data: such as integers or floating-point values
- Categorical data: data that can be divided into distinct categories or groups
- Date and Time data: data that includes a specific date and time
- Text data: unstructured data containing characters, numbers, and symbols.

This technique is simple. It checks whether the data type of the targets' attributes is the same as the attributes of the source. However, sometimes specific information about the data types is absent. Instead, the data type of the attributes is text data. Therefore, our approach tries to infer the data types of the columns from their instances.

3.1.2 Text Processing

After applying the first technique and keeping the attribute pairs with the same data type, we employ Nlp techniques to process the names of the selected attributes Vijayarani et al. [2015]. The attribute naming choices can differ a lot due to the heterogeneity of their schema. For example, "FirstName" and "Country_Code" are two attribute names from different tables that separate the words using various naming schemes. The algorithm has to account for this problem by processing the text of the attribute name to distinguish the different words inside it. When the distinction occurs, the algorithm transforms the given text utilizing the following processes:

1. Lowercasing: Converts all text to lowercase to standardize the text data.
2. Removing Punctuation: Removes all punctuation marks from the text.
3. Removing Stop Words: Removes words that frequently appear in the text but carry little meaning, such as "a," "an," "the," etc.
4. Tokenization: Divides the text into smaller units called tokens, such as words or phrases.

These text transformations are crucial to convert text data into a format easily analyzed and processed by our linguistic technique.

3.1.3 Linguistic Technique and External resource

The processed selected attribute pairs can now be used to find the possible matching pairs using a linguistic technique. Linguistic techniques utilize a string similarity measure to obtain a probability of matching the names between the source and the target attribute. Our algorithm employs cosine similarity as its string similarity measure.

Cosine similarity is a similarity measure between two non-zero vectors of an inner product space. It is defined as the cosine of the angle between the vectors. The cosine similarity ranges from -1 (entirely dissimilar) to 1 (identical).In our case, cosine values are normalized to contain values between 0 and 1. In mathematical terms, given two n-dimensional vectors of attributes, **A**, **B**, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot B}{\|A\|_2 \|B\|_2}$$

*where **A**i and **B**i are the ith components of vectors **A**, and **B** respectively* [1].

The vectors represent the text as mathematical objects in a multi-dimensional space. Each dimension in the space corresponds to a feature or attribute of the data point, and the value in each dimension represents the strength or magnitude of that feature. The process of transforming data into a vector representation is called vectorization Singh [2022]. Vectorization is a common practice in NLP, and there are a couple of different methods to be considered, including one-hot encoding, bag of words representation, term frequency-inverse document(Tfidf), and word embeddings.

[1] https://en.wikipedia.org/wiki/Cosine_similarity

- **One-hot encoding:** Each unique word in a document is assigned a binary vector with all values set to zero except for the position corresponding to the word, which is set to one.
- **Bag-of-words representation:** Each document is represented as a vector of word frequencies, with each dimension corresponding to a word in the vocabulary and the value representing the number of times that word appears in the document.
- **Term Frequency-Inverse Document Frequency (TF-IDF):** Similar to the bag-of-words representation, but the values are scaled by the inverse document frequency of each word, which down-weights words that frequently appear across many documents

These methods can be combined and fine-tuned to produce custom vector representations tailored to the specific needs of a particular text analysis task. However, these vector representations of the words do not convey the semantic relations between them, which is essential for comparing names between the attributes since the length of the string is relatively small. Another representation, called word embeddings, captures the semantic meaning of the words so that similar words are close to each other in the vector space. This is typically achieved through training a neural network on a large corpus of text. The network learns to predict surrounding words given a target word or to classify words in a particular context. A popular neural network for generating word embeddings is Word2Vec Mikolov et al. [2013].

The effectiveness of Word2Vec comes from its ability to group vectors of similar words. Given a large enough dataset, Word2Vec can estimate a word's meaning based on its occurrences in the text. These estimates yield word associations with other words in the corpus. For example, "King" and "Queen" would be similar. You can find a close approximation of word similarities when conducting algebraic operations on word embeddings. For example, the 2-dimensional embedding vector of "king" - the 2-dimensional embedding vector of "man" + the 2-dimensional embedding vector of "woman" yielded a vector that is very close to the embedding vector of "queen."

Word2Vec contains two architectures two main architectures: Skip Gram and Continuous Bag Of Words (CBOW). Skip-gram model predicts the context words given a target word. Given a target word, the model takes that word as input and predicts the surrounding words in a window around the target word. The model is trained to maximize the probability of the surrounding words given the target word. On the other hand, the CBOW model predicts a target word given its surrounding context words. Given a set of context words, the model takes the average of their word embeddings as input and predicts the target word. The model is trained to maximize the probability of the target word given the average of the context word embeddings.

Instead of generating our word embeddings, we employ as an external resource a Word2Vec pre-trained model that can generate the word embeddings without the need for training from scratch. We chose the Google News Word2Vec pre-trained model [2]. Google News Word2Vec is a pre-trained Word2Vec model trained on a large corpus of Google News text data. It has a vocabulary size of 3 million words and 300-dimensional embeddings for each word. This model is a perfect fit for finding the similarities between attribute names because it includes many words and their semantic correlation.

[2] https://code.google.com/archive/p/word2vec/

After acquiring the word embeddings of the attribute names, we compute the cosine similarity between them. Then, a threshold has to be set by the user. Finally, the correspondences that are above the given are kept.

3.2 Pre-filtering Technique In Action

In this work, we chose to test the performance of our pre-filtering Technique in action. We gathered open-sourced schema matching labeled datasets derived from current schema matching research Koutras et al. [2021], Koutras et al. [2022] to serve as our guide for evaluating our dimensionality reduction technique.

Choosing the correct threshold is essential for improving the performance of search space reduction techniques because it affects the number of attribute pairs that will be used as input for schema matching systems. Hence selecting a high threshold will remove a significant number of input pairs. However, setting a high threshold contains the probability of removing the correct correspondences. To deal with that, researchers and users of schema matching systems used a low similarity threshold for search space reduction techniques based on their intuition Rahm and Peukert [2019].

Having access to labeled datasets allows us to explore how different thresholds can affect the performance of our reduction technique and helps us suggest the correct threshold for removing the maximum number of pairs without losing the correct ones. This threshold exploration will also enable us to answer the second sub-research question (see section 1.2). Additionally, we check our methods' runtime speed and the reduction percentage of the selected against the original size of possible pairs. The performance of our method will give us insight into whether it could be a good choice for improving the efficiency of schema matching systems for reducing the search space.

In addition, we decided to benchmark the performance of our proposed method against other schema matching techniques to reduce the search space for schema matching systems. We chose to use only schema-based linguistic techniques that operate on the element level as our selection as our comparison. The choices of linguistic techniques are the following:

1. **Levenshtein distance:** This method measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. The Levenshtein distance ranges from 0 to the length of the longer string, where 0 indicates an exact match and a larger number indicates more dissimilarity.

2. **Jarro-Winkler:** This method is an extension of the Jaro distance that considers the number of matching characters at the beginning of the strings. It ranges from 0 to 1, where 1 indicates an exact match and 0 indicates no similarity

3. **Coma-Schema-Only:** Coma uses a combination of different matching techniques, including syntactic, semantic, and structural matching. We consider only the techniques used on the element level mapping following the implementation of Koutras et al. [2021]

4. **Cosine-TF-IDF:** Cosine similarity with TF-IDF (term frequency-inverse document frequency) is a technique for comparing the similarity between two documents. This approach measures the similarity between two documents as the

cosine of the angle between their vectors in a high-dimensional space. Each dimension represents a unique term in the document.

This is currently the first attempt to evaluate the performance of linguistic matching techniques to reduce the search space for possible correspondences.

3.2.1 Datasets

To evaluate schema matching techniques, we need data from different domains. The sources' diversity helps simulate a real-world scenario where data exists in different data lakes with different schemas.

- Wikidata is a dataset generated from the valentine benchmark library for schema matching Koutras et al. [2021]. It contains US musician data from Wikidata's knowledge base [3]. The labels were gathered according to four schema matching with a certain amount of actual column matches between the two datasets: Unionable: requires all columns, View-Unionable: a common subset of columns, and (Semantically)-Joinable: one common column.
- OpenData consists of tables from Canada, USA, and UK Open Data, as extracted from the authors of Nargesian et al. [2018]. The tables represent diverse sources of information, such as timetables for bus lines, car accidents, job applications, etc. These were used as training data for SiMa Koutras et al. [2022]. We used labeled data about the New York and Los Angeles municipalities for our benchmark test.

Each of these benchmarks comes with its own ground truth of matches that should hold among the columns of different datasets. Ground truth was produced automatically for datasets fabricated from the same source table and manually for pairs of columns belonging to datasets of different domains. Table 3.1 shows the number of tables that were used with the number of their labeled combinations.

	Number Of Tables	**Labeled Combinations**
Wikidata	8	4
NY data	11	28
LA data	11	42

TABLE 3.1: Number of tables and the labeled combinations for each domain

3.2.2 Evaluation

The metrics used in our benchmark measure the schema matching methods' performance. These metrics can be categorized into the following two categories.

Firstly, we consider each method's runtime speed and reduction percentage. The fastest method with the highest reduction percentages is what we are looking for in our case. The reduction percentage is acquired by using the formula below:

$$ReductionPercentage = (1 - \frac{SelectedPairs}{TotalPairs}) * 100$$

[3] https://www.wikidata.org

where SelectedPairs is the number of pairs chosen by the method and TotalPais is the maximum number of possible pairs

Secondly, we utilize classic metrics such as precision, recall, and F1-score. Precision, recall, and F1-score are metrics commonly used to evaluate the performance of a binary classifier, which assigns a binary output (e.g., "positive" or "negative") for each input sample. In our case, we handle the selected attribute pairs from the space reduction method as if they belong to the positive class and the not selected attribute pairs to the negative class.

- **Precision** refers to the proportion of true positive predictions out of all positive predictions made by the classifier. In other words, it measures the accuracy of the positive predictions. High precision means that there are few false positive predictions.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

- **Recall** (also known as sensitivity or true positive rate) refers to the proportion of true positive predictions from all actual positive samples in the data. In other words, it measures the ability of the classifier to identify all positive samples. High recall means that there are few false negatives.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

- **F1-score** is the harmonic mean of precision and recall. It measures the overall accuracy of the classifier, considering both precision and recall. The F1-score provides a single scalar value that balances precision and recall and gives a good idea of how well the method performs in terms of both.

$$F1 - Score = 2 * \frac{Percision * Recall}{Percision + Recall}$$

Precision, recall, and F1- score produce values between 0 and 1, with 1 indicating the perfect value and 0 the worst possible value. We derived the values of TP, FP, TN, and FN from the following equations:

1. $NotSelected = TotalPairs - SelectedPairs$
2. $FN = MissingPairs$
3. $TN = NotSelected - FN$
4. $TP = Labels - FN$
5. $FP = SelectedPairs - TP$

MissingPairs is the number of labeled pairs that were not selected from the method, and labels consist of the number of the actual matching pairs.

Our evaluation aims to find the threshold for each method's highest recall to avoid not selecting the correct correspondences (False Negatives). Then the methods are compared with their precision and F1-score.

3.3 Experiments

In this section, we summarize the results of the experiments mentioned in 3.2 for the selected datasets 3.2.1. First, we showcase the results of the wikidata. The goal is to perform a grid search to find the threshold value that selects the maximum number of possible pairs without removing the correct schema matching pairs for each selected method. After, we compare the performance of the methods in terms of the run-time speed, reduction percentage, and precision.

Figure 3.1 presents the number of the labeled matches that the methods missed using their best threshold, acquired from the grid search for the table combinations of wikidata. We can observe that Coma-Schema and Cosine-TfIdf did not capture all the labels. Thus, we did not use them for the next experiment.

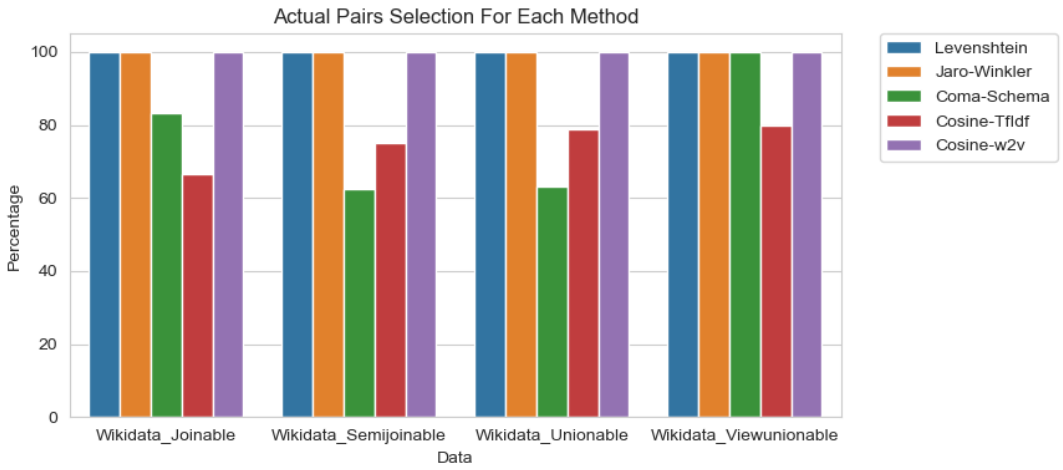

FIGURE 3.1: Number of missed correct correspondences for each method.

Table 3.2 shows the results of the other three linguistic methods (Cosine-W2V, Levenshtein, and Jaro-Winkler) for the 4 table combinations. At first glance, we notice that our method requires more run-time speed than the other two, but it reduces the maximum number of pairs in most of the cases than the other two methods.

At first glance, we notice that our method requires more run-time speed than the other two, but it reduces the maximum number of pairs in most cases than the other two methods. The average results depicted in table 3.3 prove that our Cosine-W2V is the most precise method.

Secondly, we explored the performance of the methods using the same criteria for New York and Los Angeles municipalities datasets. However, in this case, we did not consider the Coma Schema-Only and Cosine-TfIdf methods since they did not capture all the labeled matches in the previous exploration. New York Data includes 28 table combinations, and the Los Angeles Data 42. The tables 3.4 and 3.5 showcase the average results of our experiments. The detailed results for each method can be seen in the appendix A.

We can observe that the Levenshtein method did not capture the labeled pairs for all the cases in New York and Los Angeles table combinations since the average recall is 0.93 and 0.74, respectively. The average recall value below one means that there were cases where the method assigned actual pairs in the Negative class (False Negatives). Additionally, for the new york table combination, the Cosine-W2V has an average runtime speed of 0.09585 ms and an average precision of 0.331768 of

Method: Cosine -W2V					
Tables (Target - Source)	**Total Pairs**	**Selected Pairs**	**Actual Pairs**	**Threshold**	**Time**
Wikidata Joinable	169	26	6	0.4	0.181
Wikidata Semijoinable	196	100	8	0.2	0.192
Wikidata Unionable	400	319	19	0.1	0.404
Wikidata Viewunionable	169	22	5	0.5	0.170
Method: Jaro-Winkler					
Tables (Target - Source)	**Total Pairs**	**Selected Pairs**	**Actual Pairs**	**Threshold**	**Time**
Wikidata Joinable	169	27	6	0.6	0.118
Wikidata Semijoinable	196	175	8	0.4	0.135
Wikidata Unionable	400	389	19	0.3	0.275
Wikidata Viewunionable	169	169	5	0.6	0.115
Method: Levenshtein					
Tables (Target - Source)	**Total Pairs**	**Selected Pairs**	**Actual Pairs**	**Threshold**	**Time**
Wikidata Joinable	169	86	6	12	0.113
Wikidata Semijoinable	196	99	8	11	0.132
Wikidata Unionable	400	335	19	14	0.266
Wikidata Viewunionable	169	21	5	9	0.121

TABLE 3.2: Wikidata Threshold Exploration Results

	Wikidata				
Method	**Time**	**Reduction %**	**Recall**	**Precision**	**F1-Score**
Levenshtein	0.158514	51%	1.0	0.111347	0.192982
Jaro-Winkler	0.161698	45%	1.0	0.125491	0.214176
Cosine-W2V	0.237367	60%	1.0	0.149401	0.251486

TABLE 3.3: Average results of the wikidatas' table combinations

New York Data					
Method	**Time**	**Reduction %**	**Recall**	**Precision**	**F1-Score**
Levenshtein	0.069899	63%	0.93	0.240446	0.307500
Jaro-Winkler	0.073362	41%	1.0	0.210041	0.270083
Cosine-W2V	0.097885	66%	1.0	0.323777	0.419710

TABLE 3.4: Average results of the New Table table combinations

Los Angeles Data					
Method	**Time**	**Reduction %**	**Recall**	**Precision**	**F1-Score**
Levenshtein	0.068388	75%	0.74	0.256561	0.347213
Jaro-Winkler	0.071954	45%	1.0	0.190282	0.244723
Cosine-W2V	0.095327	64%	1.0	0.331768	0.418983

TABLE 3.5: Average results of the Los Angeles table pairs

the total attribute pairs, compared with the Jaro-Winkler method that achieved an average runtime speed of 0.071954 ms and precision of 0.190282. The Los Angeles table combinations experiment follows the same pattern, where the Cosine-W2V requires more runtime speed and achieves higher precision than the Jaro-Winkler method. The results for the Los Angeles case can be seen in table 3.5.

Figure 3.2, and figure 3.3 show the distribution of the threshold values for the

two remaining methods for the New York, and the Los Angeles table combinations, respectively. We can observe that the minimum threshold is 0 for the Cosine-w2v method and the Jaro-Winkler method for both table combinations. The minimum value for each method is the value that does not remove actual pairs from their selection for all cases for both methods.

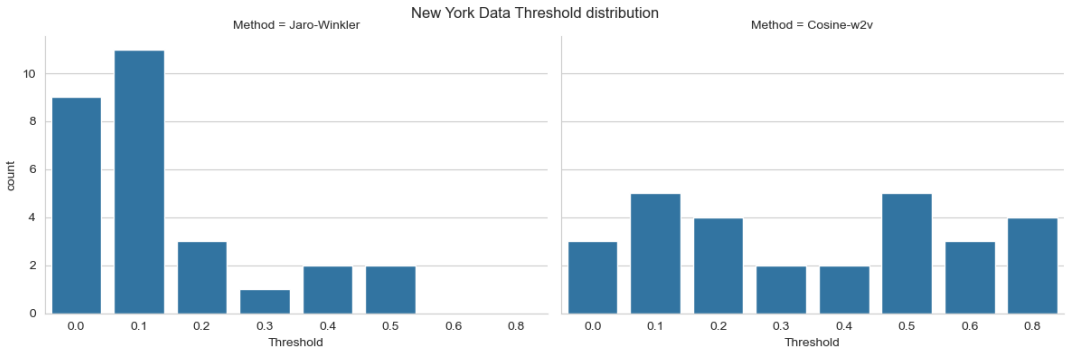

FIGURE 3.2: The New York Data threshold distribution of the two methods

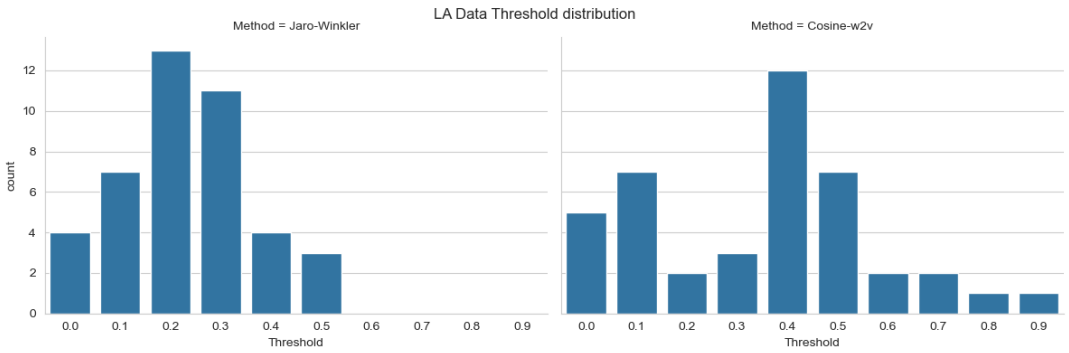

FIGURE 3.3: The Los Angeles Data threshold distribution of the two methods

3.3.1 Constraint-based technique experiments

Here we conduct the final experiment to see how the linguistic approaches perform with the addition of the constraint-based technique. First, we apply the constraint method to select the pairs that hold the same data type, and then we utilize the linguistic method with the minimum threshold we acquired from the previous exploration.

Table 3.6 compares the results of cosine-w2v without using the constraint technique and this technique for the cases where cosine-w2w required the minimum threshold from the previous experiments. The table shows a high increase in reduction when the constraint technique is applied.

Table Combinations	Reduction1	Reduction2	Precision1	Precision2
wikidata unionable	20%	36%	0.059561	0.073643
nydata(city_record_online vs film_permits)	3%	47%	0.062857	0.117021
nydata(housing vs film_permits)	2.5%	43%	0.025641	0.084507
nydata(agency_spending vs film_permits)	1%	45%	0.031579	0.109091
ladata(asset_inventory vs city_compaign)	3.6%	35%	0.041667	0.062069
ladata(city_reg_lobbyists vs asset_inventory)	5.8%	40%	0.052133	0.082090
ladata(clients_registered_lob_firms vs asset_invenotry)	2.9%	38%	0.019802	0.031496
ladata(payment_client_lob_firms vs asset_inventory)	10%	40%	0.070175	0.105263
ladata(city_proj_agencies_lob_firms vs asset_inventory)	8.3%	32%	0.045455	0.061538

TABLE 3.6: Results of the table combinations that required the minimum threshold for each case. 1) refers to Cosine-w2v, and 2)Cosine-w2v with the constraint reduction technique

Finally, table 3.7 showcases the average results for the New York and the Los Angeles table combinations for the Cosine-w2v and the Jaro-Winkler with the minimum threshold and the usage of the constraint technique. The table shows that both linguistic methods perform similarly with the minimum threshold value.

	New York		Los Angeles	
	Cosine-w2v	**Jaro-Winkler**	**Cosine-w2v**	**Jaro-Winkler**
Avg Time	4.601300	4.516401	0.711806	0.690157
Avg Reduction	41%	42%	31%	30.5%
Avg Recall	1.0	1.0	1.0	1.0
Avg Precision	0.087285	0.087547	0.065783	0.065366
Avg. F1-score	0.149865	0.150834	0.118623	0.117904

TABLE 3.7: asdasd

Our experiments have shown that our proposed technique effectively reduces the search space of attribute matches while retaining labeled matches. This approach is efficient and suitable for various schema matching scenarios. However, we have also observed that Jaro-Winkler and Cosine W2V similarity perform similarly when using the minimum threshold. Nonetheless, we found that Jaro-Winkler similarity generally requires a lower threshold than Cosine W2V, while cosine W2V can perform better when a higher threshold is used. Therefore, we suggest that users choose between these two similarity measures for this evaluation stage until further testing is performed.

Chapter 4

Crowdie: a crowdsourcing platform for schema matching

This chapter outlines the decision-making process that led to the creation of Crowdie [1], our prototype crowdsourcing tool for generating labeled data for schema-matching tasks. We then present the system architecture of Crowdie in section 4.1, which also incorporates the user interface design. Section 4.2 details the task design choices made for Crowdie.In section 4.3, we discuss the quality control measures implemented in Crowdie. In section 4.4, we present the experiments we used to evaluate the platform and discuss the results in section 4.5.

[1] https://github.com/delftdata/msc-crowdie

4.1 System Architecture

As described in section 2.3, crowdsourcing is a practical approach to enhance the effectiveness of schema matching by utilizing human intuition to verify the results of schema matching. In our work, we leverage crowdsourcing to generate labeled data that can be utilized to evaluate and improve the effectiveness of schema matching systems. To this end, we introduce Crowdie, a prototype crowdsourcing tool designed for generating labeled data for schema-matching tasks. Crowdie is designed for two main schema matching scenarios: joinable and unionable column search. Joinable relationships refer to two tables that can be joined on a shared attribute or set of attributes. In comparison, unionable relationships refer to two tables that can be merged, with columns from one table appended to another.

Crowdie integrates automatic search space reduction techniques, as discussed in chapter 3, to generate tasks and microtasks for human verification. The tool takes a collection of schemata, each containing a set of attributes, as input (as shown in Figure 4.1). The administrator uploads the collection of tables to Crowdie. Given this collection, Crowdie uses the *Pre-Filtering module* to generate all the reduced possible attribute matchings. The Pre-Filtering Module first finds all possible combinations of the given tables and then generates all possible attribute matches for each pair using the reduced space reduction technique. Next, the *Task Creation Module* creates the corresponding crowdsourcing tasks with their associated microtasks for the reduced pairs. These tasks are then published for the crowd workers to answer via the *Crowdie UI*. Once all the tasks have been completed, and the answers have been received from the crowd, Crowdie uses the *Aggregation Module* to combine the results so that the admin can take them.

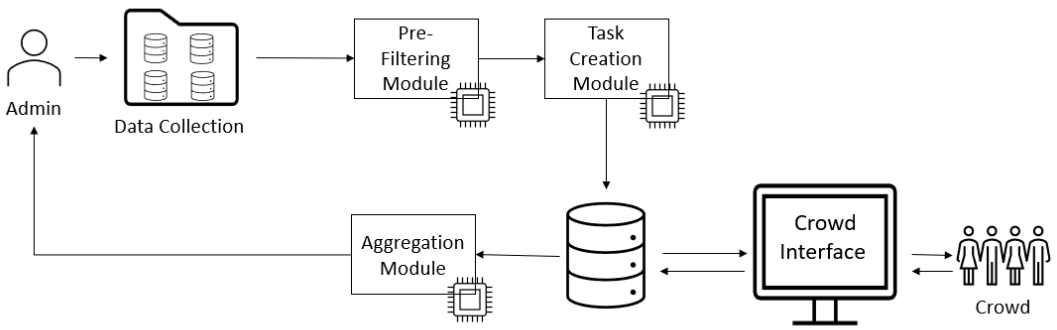

FIGURE 4.1: Crowdie: System Architecture.

4.1.1 Pre-Filtering Module

The Pre Filtering module is responsible for processing collections of tables that the administrator provides. It starts by identifying all possible combinations of tables, and for each combination, it finds all column pairs between them. The module then applies a pre-filtering algorithm to reduce the number of column pairs based on a threshold specified by the administrator. This algorithm ensures that only the most relevant column pairs are considered for further processing.

Once the reduced column pairs are identified, the module samples rows for each pair. If any of the columns within a pair have NaN values, the pair is removed

from consideration. After completing its tasks, the Pre Filtering module forwards its results to the Task Creation module. The results include the reduced column pairs and the sampled rows for each pair. The Task Creation module uses these results to create tasks and microtasks for workers to complete.

Overall, the Pre Filtering module is critical in preparing the data for downstream processing by identifying the most important column pairs and reducing the number of pairs.

4.1.2 Task Creation Module

In Crowdie, the Task Creation Module is responsible for creating tasks and microtasks from the results of the Pre-Filtering Module. A task in Crowdie refers to the Result Set (RS), which contains all possible column combinations between two tables, as in Zhang et al. [2014]. However, Crowdie also contains a sample of the rows for each column in the result set. For example, if we have two tables with columns column1.1 with rows1.1, column1.2 with rows1.2, column2.1 with rows2.1, and column2.2 with rows2.2, the RS would be:

$$RS = \{(column1.1[rows1.1], column2.1[rows2.1]), (column1.1[rows1.1], column2.2[rows2.2]), (column1.2[rows1.2], column2.1[rows[2.1]])(column1.2[rows1.2], column2.2[rows2.2])\}$$

This RS is then used as the basis for creating microtasks. A microtask in Crowdie is a specific task instance containing part of the columns presented to workers through the user interface. Workers can focus on individual comparisons and provide more accurate results by breaking down the task into smaller microtasks.

After the Task Creation Module generates the tasks and microtasks, it assigns unique keys to each task and its associated microtasks before storing them in a relational database. This approach ensures easy access and management of the tasks and microtasks. Workers can efficiently complete their assigned tasks, and administrators can monitor progress and make adjustments as necessary. With unique keys, the Task Creation Module can track which microtasks have been completed and which are still pending. The task creation module answers the first part of our 3rd sub-research question, "What is considered a task for schema matching?".

4.1.3 Crowdie's User Interface

The design of Crowdie's user interface was created with simplicity and consistency in mind. The goal was to make it easy for users to navigate the site and complete tasks, by providing a clean and uncluttered layout. We used a consistent color scheme, font selection, and imagery throughout the website to achieve this. The three main pages in the Crowdie UI are the Welcome, About, and Tasks pages.

The Welcome page is the first page users see when they visit the Crowdie website (see figure 4.2). It introduces the site's purpose, and users must sign in as crowd workers to participate in the labeling process. The sign-in form consists of two input fields for the user to enter their username and select whether they have a computer science background by choosing "Yes" or "No" 4.3. After submitting the form, the platform assigns a unique key for the current worker, which is essential in keeping track of the tasks they have completed. The unique key also identifies the user and allows them to navigate to the other pages on the website.

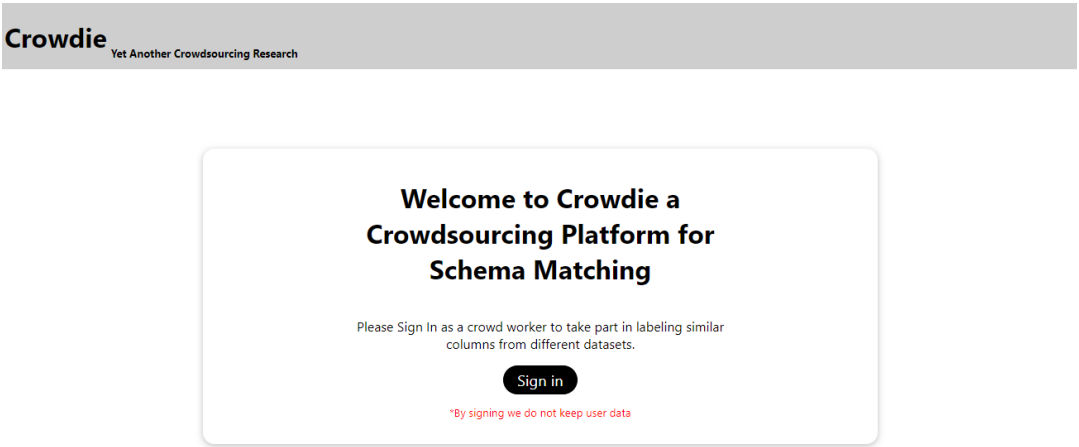

FIGURE 4.2: Crowdie: Welcome Page

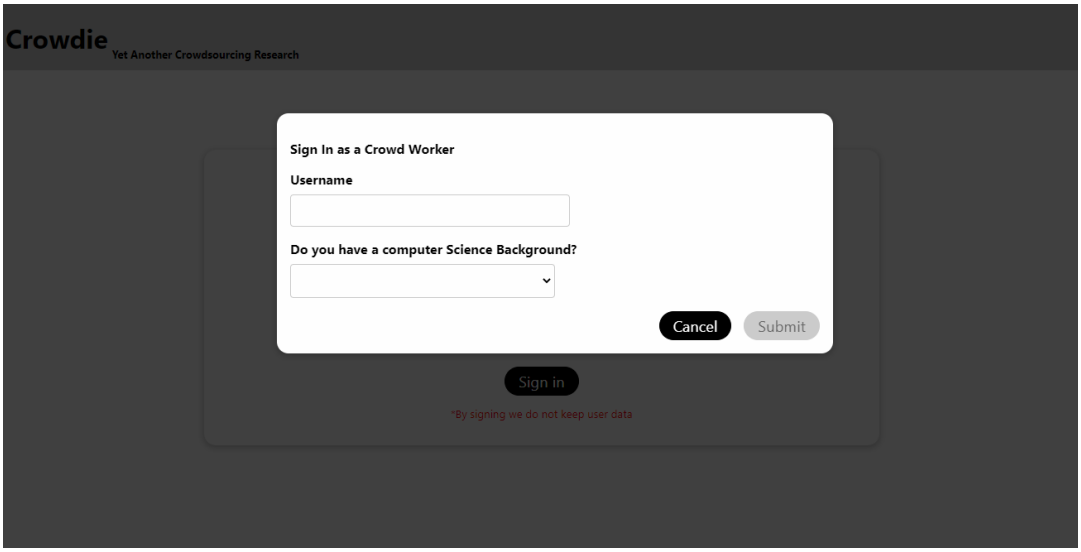

FIGURE 4.3: Crowdie: Sign In

The About page on Crowdie 4.4 is where workers can find detailed information about the platform's mission and purpose. This page explains that Crowdie is a platform designed to generate labeled datasets for schema matching, a crucial task in data management. Workers can learn about the importance of this work and what is required to ensure accurate and effective results. The page also offers a detailed task description, outlining what workers can expect from their labeling tasks. Additionally, it provides background information about Schema Matching and joinable and unionable relationships (see figure 4.5).

The Tasks page is where the labeling process occurs for workers on the Crowdie platform. When workers access this page, they can see the current number of available tasks. The page briefly describes how to complete the microtasks, which involve determining whether two columns are similar. To start labeling, workers must click the "Load Microtasks" button, which loads ten microtasks for the current task 4.6. Workers can answer each microtask by selecting either "Yes" or "No" to indicate whether they believe the columns are similar. After answering all ten microtasks, workers can submit their answers, which are stored on the platform for later use.

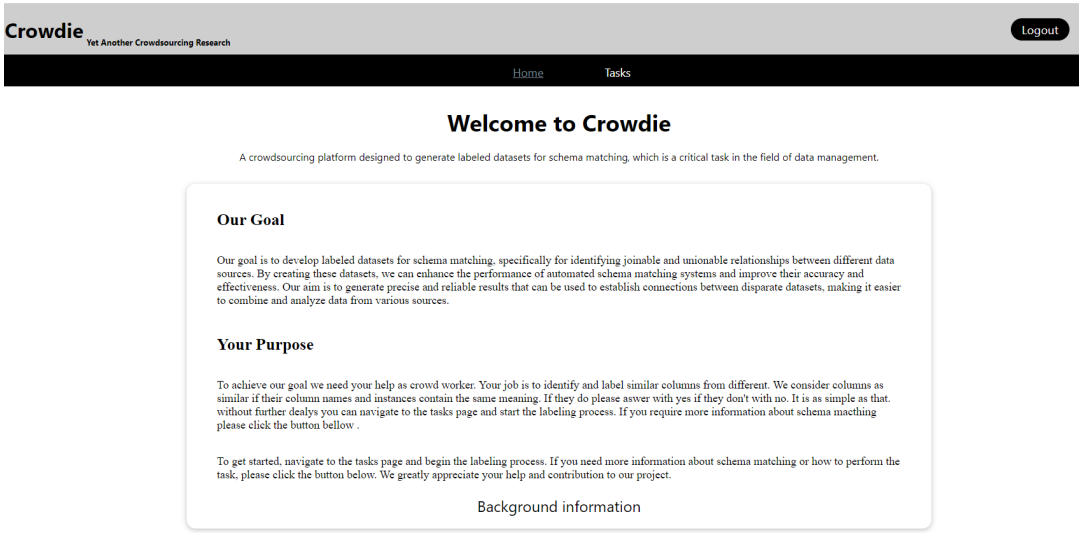

FIGURE 4.4: Crowdie: About Page

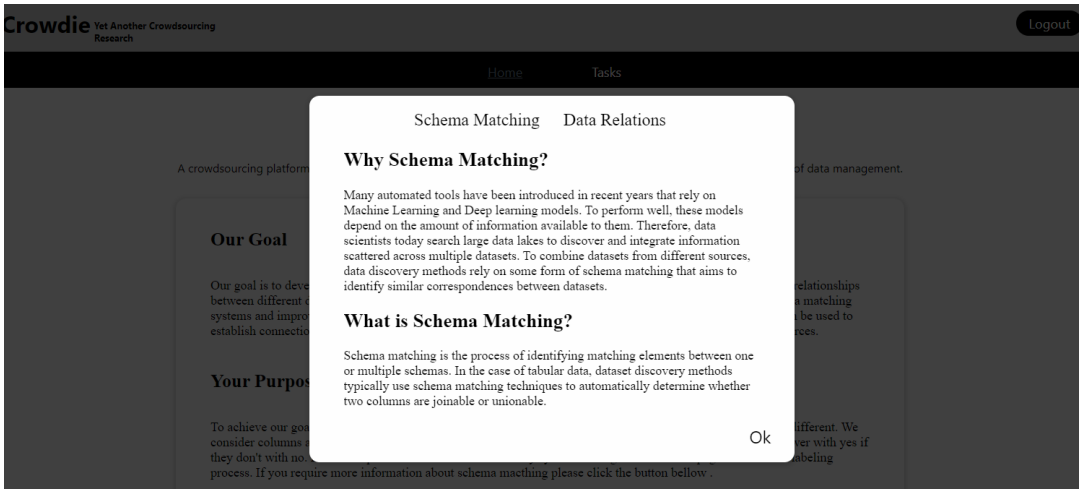

FIGURE 4.5: Example microtask: Workers are asked to match columns between two tables.

The design choices of the task is discussed in section 4.2 When the worker submits the first batch of the microtasks, the platform allows him to continue labeling more microtasks by pressing the "Continue" button 4.7.

The simplicity and straightforwardness of the labeling process on the Tasks page are designed to make it easy for workers to complete their tasks accurately and efficiently. Clear instructions and easy-to-use features allow workers to focus on the task and produce high-quality labeled datasets for schema matching.

In summary, Crowdie's UI design prioritizes simplicity and consistency, focusing on ease of use for workers completing labeling tasks. The Welcome page requires users to sign in as crowd workers to access the platform, while the About page provides detailed information about the platform's mission and purpose. The Tasks page is where the labeling process occurs, with clear instructions and easy-to-use features to ensure accuracy and efficiency.

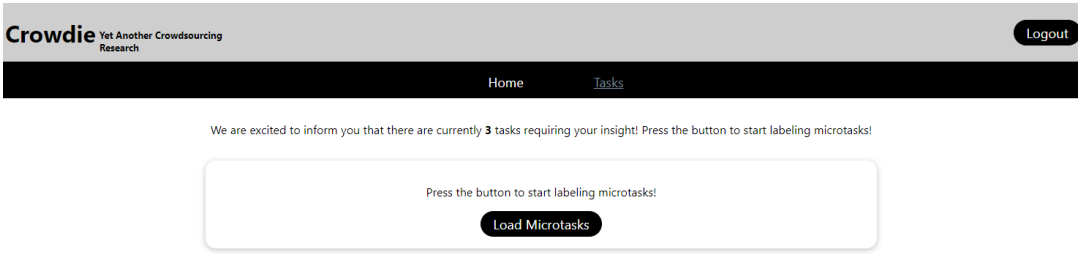

FIGURE 4.6: Tasks Page: Available Tasks and Load Microtasks

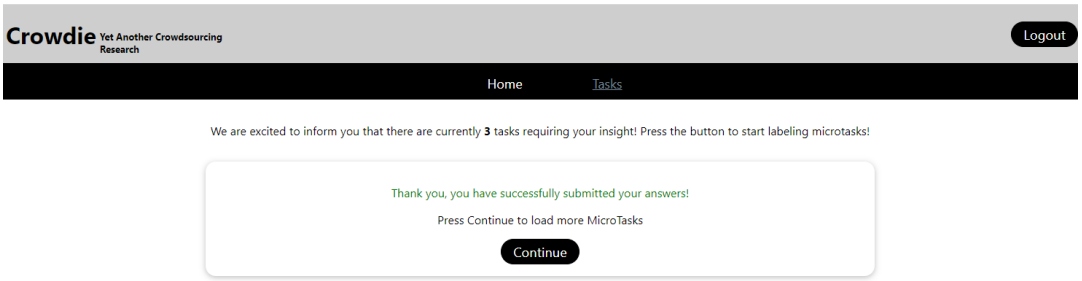

FIGURE 4.7: Tasks Page: Success and Continue

4.1.4 Aggregation Module

The aggregation module plays a crucial role in managing the workflow of crowdsourcing tasks. It assigns available tasks and microtasks to workers based on their availability. The module also monitors the workers' progress to ensure the completion of the tasks within the specified quality standards. If any task or microtask is incomplete, the module reassigns it to another worker.

Once all of the tasks have been completed, the aggregation module collects the responses from each worker and aggregates them to generate a final output for the administrator. Additional information on when a task is considered completed, and the aggregation method can be found in section 4.3

4.2 Task Design

Task design usually refers to the design of the user interface for the task, determining the type and requirements of the task Zheng et al. [2011]. In the case of schema matching the type of the task is binary. Typically it requires an answer on whether two columns are a match.

Crowdsourcing has been commonly used as a means of evaluating the results of schema matching systems Zhang et al. [2014]. However, the task design used in previous studies was limited to asking workers whether a specific column from one table matches a specific column from another (e.g., "Does table1.column1 match with table2.column2?").

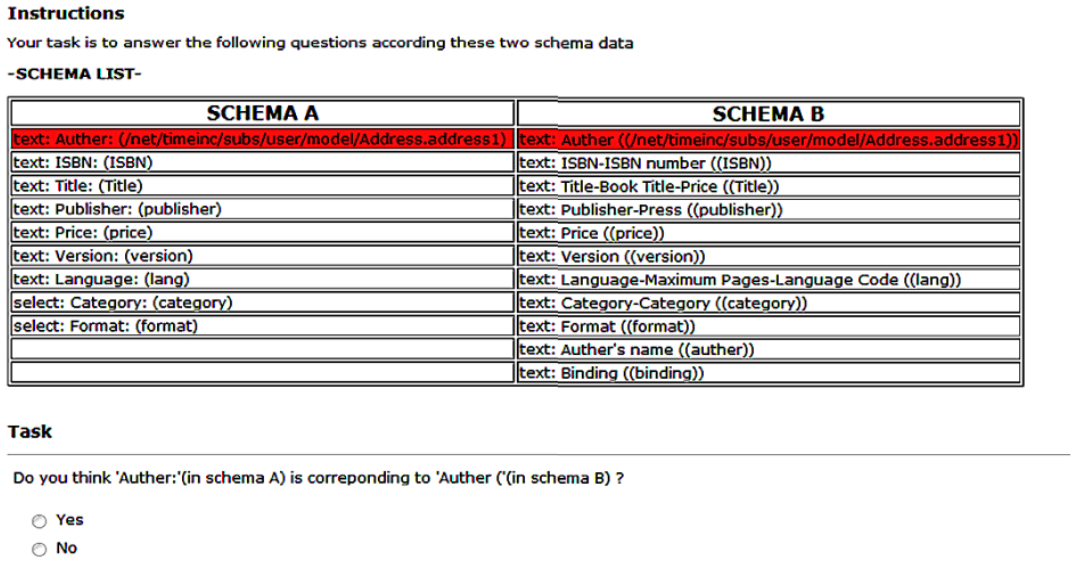

FIGURE 4.8: Task Design of CrowdMatcher Zhang et al. [2014]

Figure 4.8, depicts the task design used in CrowdMatcher Zhang et al. [2014]. While this design is simple and easy to understand, it does not provide additional contextual information Hung et al. [2013], such as the instances of the schemata, which could impact the accuracy of the results obtained.

To address this limitation, alternative task designs could provide workers with more contextual information. In addition to the names of the tables and columns being matched, it could also provide the content of the columns. Providing more detailed examples of how workers should approach the task can also improve the results' quality. Figure 4.9 shows an example of a microtask in Crowdie.

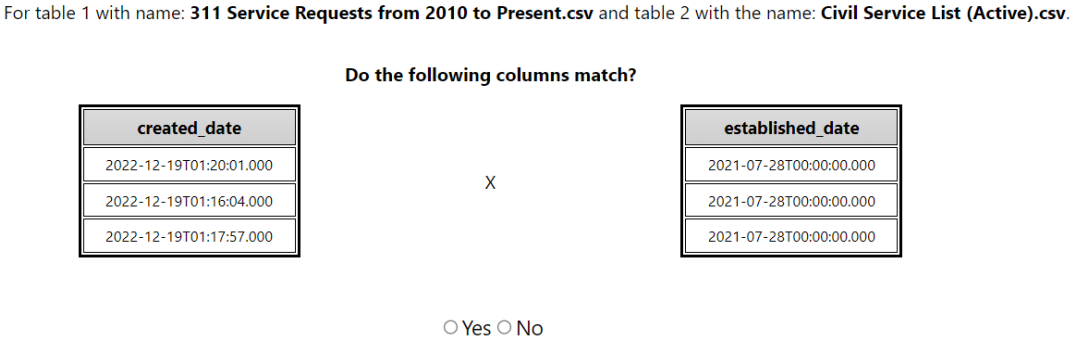

FIGURE 4.9: Example microtask: Workers are asked to match columns between two tables.

The workers of Crowdie are initially presented with the names of the tables currently being searched. This is also one of the available tasks. To do this, workers are asked "Do the following columns match?" and shown a table format with the column names and contents, which are part of the above tables. Workers must submit their answer by selecting one of two radio buttons, indicating either "Yes" or "No". For example, in figure 4.9, the worker is asked whether the column with the name "created date" from the table "311 Service Requests from 2010 to Present.csv" matches with a column "established date" from the table "Civil Service List (Active).csv".

The number of microtasks assigned to each task is determined by the number of column pairs identified by the pre-filtering module. For instance, one task may have 100 microtasks while another may only have 40, depending on the number of column pairs. Displaying all the microtasks at once can be overwhelming for the workers, so we have decided to display a limited number of microtasks to improve their focus.

The administrator sets the number of microtasks displayed at any given time, with a maximum limit of 10. Workers can concentrate on specific microtasks without feeling overwhelmed or distracted. This aims aim to improve the accuracy and efficiency of the crowdsourcing process. Figure 4.10 displays a list of microtasks, with each group comprising a set number of 2.

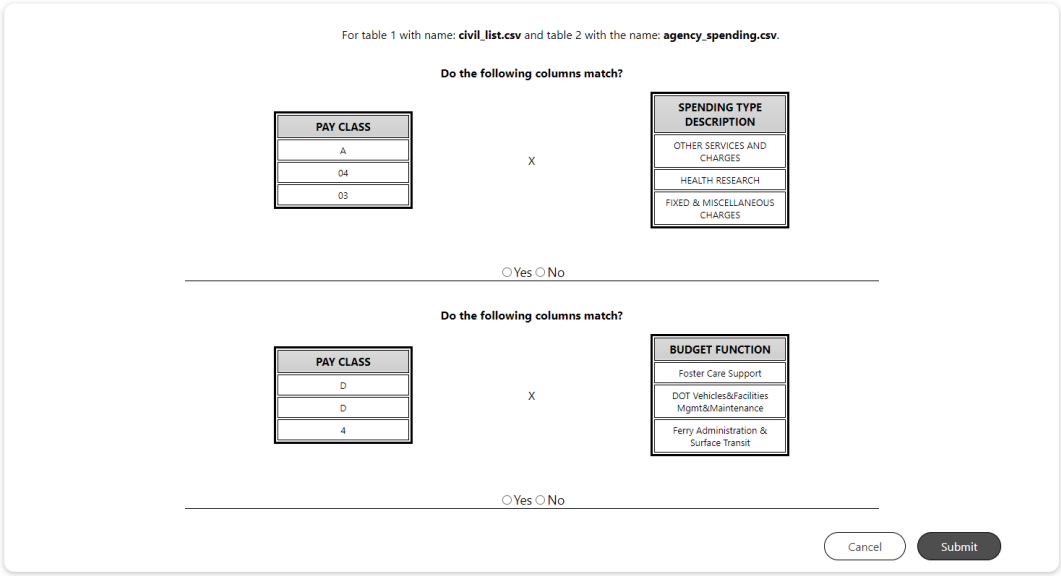

FIGURE 4.10: Example microtask: Workers are asked to match columns between two tables.

CrowdIE's task design aims to provide more contextual information to workers without overwhelming them with unnecessary details by keeping a simplistic task design Finnerty et al. [2013].

4.3 Quality Control

Quality control in crowdsourcing aims to ensure that the results are accurate and reliable Daniel et al. [2018]. In Crowdie, a task is considered complete only when at least three different workers have provided their answers to each microtask, ensuring that multiple workers have reviewed the same data to reduce the risk of errors or biases. We suggest using the minimum number of answers required for each microtask to keep the cost low. However, the administrator can increase the number of answers for each microtasks. Furthermore, the platform prohibits the same worker from answering a microtask more than once to maintain fairness.

Once all the microtasks have been completed, the aggregation module uses majority voting to determine the correct label when two workers have agreed on an answer. This approach helps to minimize errors and biases and ensure the highest possible accuracy in the results. Finally, the administrator can then review and label the data, providing an additional layer of quality control to ensure that the results

meet the desired standards. Overall, Crowie's approach to quality control in crowdsourcing demonstrates a commitment to ensuring the results' accuracy, reliability, and fairness.

4.4 Evaluation

To assess the effectiveness of Crowdie, we will need to recruit individuals to participate as workers on our prototype platform. Finding participants can be quite time-consuming, as it involves advertising the purpose of our research to relevant online forums and configuring incentives to motivate individuals to complete the required tasks. Alternatively, we could integrate the user interface with a crowdsourcing platform like Amazon Mechanical Turk to obtain workers. However, due to the time constraints of this project and the fact that Crowdie is still in the prototype stage, we have decided to conduct a pilot study by inviting individuals from our social networks.

We successfully recruited 10 participants to assist us in our pilot study. Table 4.1 shows the number of participants with their background knowledge, and four participants had a computer science background. The workers with an experience in computer science are labeled as experts in this pilot study due to their knowledge of what a schema is and their understanding of schema matching procedures.

Number of Workers	Computer Science Background
4	Yes
6	No

TABLE 4.1: Number of people who participated in the pilot study with their knowledge background in computer science.

In this study, we assigned each worker a task involving labeled data, which we used to evaluate the effectiveness of our search space reduction technique as described in section 3.2.1. The main goal was to assess the ability of the workers to identify correct matches in the given data. We designed a set of tasks, represented in Table 4.2, where each row corresponds to a unique task assigned to a single worker. The "Pairs" column in the table indicates the number of microtasks generated by the pre-filtering module. In contrast, the "Total" column represents the total number of column combinations or possible microtasks without applying the pre-filtering module.

Table1	Table2	Pairs	Total
election_list.csv	city_campaign_contribs.csv	63	98
election_list.csv	clients_registered_lob_firms.csv	62	91
city_reg_lobbyist.csv	city_proj_agencies_lob_firms.csv	51	84
city_reg_lobbyist.csv	payment_client_lob_firms.csv	31	56
payment_client_lob_firms.csv	asset_inventory.csv	38	64
recent_contract_awards.csv	agency_spending.csv	57	96
recent_contract_awards.csv	capital_projects1.csv	24	60
housing.csv	capital_projects2.csv	32	40
evictions.csv	capital_projects1.csv	25	60
evictions.csv	capital_projects2.csv	42	48

TABLE 4.2: The tasks that were used in the pilot study.

After completing their assigned task, we asked each worker to provide feedback on the platform and share any insights they gained. We posed several open-ended questions to the crowd workers to obtain valuable feedback on the design and labeling process. Specifically, we asked: *"What are your thoughts on the design of the user interface?"*, *"Can you describe your experience with labeling the columns?"*, *"Did you understand the goal of the task?"* and *"Can you provide specific examples of labeling that you found difficult?"*. These questions were designed to elicit detailed and honest feedback and encourage participants to provide specific examples where necessary. By asking these focused questions, we aim to gain insight into the platform's usability and identify areas for improvement.

4.5 Results

In this section, we present a summary of the findings from our pilot study. Table 4.3 presents the labeling outcomes, indicating the number of "yes" responses given by each worker for their assigned task and the actual number of labels for that particular task. Furthermore, Table 4.4 shows the workers' microtasks and whether they were correctly labeled.

Regrettably, not all workers could correctly identify all labels, except for workers 2 and 8, who accurately identified all the labels for their respective tasks. Upon a preliminary analysis of the results, we observed that workers with a computer science background could identify more labels correctly than those without such a background. Furthermore, we discovered that worker 6 labeled a task that contained identical column names but did not belong to the label set for that particular task. Therefore, this labeling was incorrect and should be disregarded.

	Background	No Task	Answer	Labels
worker 1	No	1	0	1
worker 2	Yes	2	1	1
worker 3	No	3	1	5
worker 4	No	4	1	5
worker 5	Yes	5	2	3
worker 6	No	6	1	2
worker 7	No	7	0	2
worker 8	Yes	8	1	1
worker 9	No	9	0	1
worker 10	Yes	10	0	1

TABLE 4.3: Number of answers each worker provided for their tasks with the number for labels of that particular task.

Furthermore, after completing the task, all the workers provided feedback for the open-ended questions described in section 3. A summary of their responses to each question is provided below:

1. **What are your thoughts on the design of the user interface?**
   All workers found the UI understandable and easy to navigate, and no notable comments were provided.

2. **Can you describe your experience with labeling the columns?**
   All workers agreed the labeling process was easy, clear, and not cumbersome.

Task No	Col_1	Col_2	Correct Answer
2	POLLING PLACE ZIP	Client Zip	Yes
3	Lobbying Firm	Lobbying Firm	Yes
4	Lobbying Firm	Lobbying Firm	Yes
5	Client Last Name	Name	No
5	Client Last Name	Owner	Yes
6	AgencyName	AGENCY NAME	No
8	Project Name	Project Name	Yes

TABLE 4.4: Shows the column names of the labeled microtasks from the workers and whether they are correctly labeled

3. **Did you understand the goal of the task?**
   Some workers provided interesting comments regarding the task description being too detailed. One worker mentioned they did not understand how to label the columns, specifically whether they should be labeled based on their exact meaning or the content of the columns. This worker pointed out that "Do the following match?" was ambiguous.

4. **Can you provide specific labeling examples that you found difficult?**

   Worker 2 observed that a microtask contained the fields 'Lobbyist Last Name' and 'Client Last Name', which may appear to have similar meanings, but their content was distinct. On the other hand, Worker 6 reported a task that included URLs and expressed uncertainty about whether it referred to the same concept as the microtask in question, which involved the fields 'Lobbying Firm Quarterly Report' and 'URL'. Unfortunately, we did not receive any other comments.

One of the key insights we gained from the pilot study is the need to improve our microtask design. To achieve this, we will modify our question prompt from "Do the following columns match" to "Are these two columns similar," which should reduce worker confusion. Additionally, we will provide a brief task description before workers start each new task, along with a help button that will be displayed during task generation. These changes will make it easier for workers to understand the task goals and perform their tasks effectively.

Moreover, our pilot study has shown that crowdsourcing is an effective method for obtaining labeled data for schema matching, with only a limited number of incorrect responses received. We are confident that with more experiments, a larger pool of workers, and design refinement, Crowdie can become a successful crowdsourcing platform for generating high-quality labeled data for schema matching tasks.

Chapter 5

Conclusion

5.1 Summary

This thesis addresses the challenge of generating labeled data for schema matching. Initially, we aimed to explore schema matching techniques that could serve as a pre-processing step to minimize the search space for schema matching tools while answering our first sub-research question:

How to create a pre-filtering schema matching technique for search space re- duction?
We have researched various low-cost schema matching techniques to develop a pre-filtering algorithm that can effectively reduce the search space by eliminating irrelevant correspondences Bernstein et al. [2011]. Our approach involves a combination of constrained and linguistic matching methods. We have also proposed a new linguistic technique that utilizes word embeddings from the Google News 300 pre-trained W2V model. This technique could be applied as a preprocessing step for any schema matching scenario to streamline the search for potential matches and enhance the efficiency of the schema matching process (see **??**).

However, implementing the pre-filtering method necessitated setting a threshold value, which raised our second sub-research question.

What is the optimal threshold value for search space reduction techniques?
Typically, pre-filtering techniques require a threshold value to identify and remove unnecessary columns. However, determining the optimal threshold value traditionally relied on the user's intuition. In our case, we have developed a pre-filtering algorithm that has been benchmarked against similar techniques for reducing the search space in schema matching. To determine the best threshold value, we performed a grid search. We found that a threshold value of 0 would qualify for all cases without eliminating any actual matches (see section 3.3).

To obtain labeled data, it is often necessary to employ crowdsourcing solutions. However, developing a crowdsourcing system requires careful consideration of various factors. This prompted our investigation into the requirements for creating such a system, leading to our third sub-research question.

What is considered a task, and how to design a task for schema matching?
Designing tasks is critical in developing any crowdsourcing application, including schema matching. To enable users to identify similar columns between two schemata, we first defined what constituted a task and microtask for our application, drawing on prior research on interface design for schema matching verification and crowd interfaces Fan et al. [2014]. In our proposal, presented in section 4.2, we put forward a novel design for schema matching that is simple and user-friendly, in line with the principles outlined in Finnerty et al. [2013]. We also included table and column names and part of the content of the columns to make the task more comprehensive and informative. By incorporating these features, we believe our proposed task design will facilitate the efficient and accurate identification of matching schema columns and improve the overall effectiveness of the schema matching process.

Ensuring the quality of the crowdsourcing pipeline is another crucial requirement, which led to our third sub-research question.

How to aggregate the results from the crowd?
In section 4.3, we described how we ensured the quality of the labeling process by using the majority voting method as our aggregation function. Majority voting is a well-established and reliable technique for aggregating results from a crowd. This method could reduce the impact of individual errors or biases and obtain a more accurate result. To further improve the reliability of the labeling process, each microtask requires answers from different workers.

Through our investigations into the aforementioned sub-research questions, we developed a prototype crowdsourcing platform for schema matching called Crowdie. This platform provides a solution to the main research question of this thesis:

How to create a system architecture for a crowdsourcing platform that generates labeled datasets for schema matching systems?
In chapter 4, we presented the system architecture of Crowdie, our prototype crowdsourcing platform for schema matching. We believe that our architecture is a promising solution for generating labeled data, as it incorporates a pre-filtering algorithm that reduces the number of possible correspondences and improves the platform's efficiency while minimizing the cost of crowdsourcing. Furthermore, our platform utilizes a simple yet effective task design for workers to label, ensuring high-quality labeled data. Overall, we are confident that Crowdie can help address the challenge of generating labeled data for schema matching and advance research in this field.

5.2 Discussion Future Directions

In this section, we propose several improvements that can be made in the future to reduce the search space and enhance the crowd platform.

5.2.1 Search Space Reduction in Schema Matching

To increase confidence in our methods, we suggest collecting more data to evaluate our methods further. Additionally, we could create our word embeddings from various schemas instead of relying solely on pre-trained word2vec models. We could also apply linguistic techniques on instances, not just column names, to improve the effectiveness of the pre-filtering algorithm.

5.2.2 Crowdie

Crowdie is still in the prototype state, and from the first pilot study, we observed that some users had difficulty understanding the task description. Therefore, we aim to improve the task description page to ensure it conveys the task's objective clearly so that all users can understand it without a background in computer science. Another improvement is to change the question of the micro-tasks from "Do the following columns match?" to "Are these two columns similar?" to prevent users from searching for exact matches.

We suggest acquiring more people to participate in a second evaluation with labeled datasets to evaluate Crowdies' performance further. We could also create a new dataset for a specific domain to see how Crowdies' performance compares. Additionally, users could provide more feedback on their experience with the platform. We recommend providing a user interface with a dashboard for the administrator, allowing anyone to use the app and check the task processing at any time.

Overall, these improvements could enhance the performance and usability of the schema matching and crowd platform, ultimately leading to better outcomes for users.

Appendix A

Appendix A

A.1 Part of the results for grid search the new york and la data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time	Reduction
56	Cosine-w2v	civil_list.csv	evictions.csv	48.0	8.0	2	0.0	0.4	0.049945	83.333333
57	Cosine-w2v	civil_list.csv	city_record_online.csv	60.0	6.0	2	0.0	0.5	0.056004	90.000000
58	Cosine-w2v	civil_list.csv	agency_spending.csv	32.0	3.0	2	0.0	0.5	0.030779	90.625000
59	Cosine-w2v	city_record_online.csv	film_permits.csv	180.0	175.0	11	0.0	0.0	0.174087	2.777778
60	Cosine-w2v	city_record_online.csv	recent_contract_awards.csv	180.0	29.0	13	0.0	0.5	0.179825	83.888889
61	Cosine-w2v	city_record_online.csv	agency_spending.csv	120.0	6.0	3	0.0	0.6	0.122137	95.000000
62	Cosine-w2v	city_record_online.csv	capital_projects1.csv	75.0	60.0	9	0.0	0.1	0.073063	20.000000
63	Cosine-w2v	city_record_online.csv	evictions.csv	180.0	70.0	5	0.0	0.2	0.169693	61.111111
64	Cosine-w2v	city_record_online.csv	housing.csv	150.0	5.0	4	0.0	0.6	0.145301	96.666667
65	Cosine-w2v	evictions.csv	film_permits.csv	144.0	97.0	5	0.0	0.1	0.136639	32.638889
66	Cosine-w2v	evictions.csv	recent_contract_awards.csv	144.0	56.0	4	0.0	0.2	0.130576	61.111111
67	Cosine-w2v	evictions.csv	city_record_online.csv	180.0	25.0	2	0.0	0.4	0.161899	86.111111
68	Cosine-w2v	evictions.csv	housing.csv	120.0	14.0	3	0.0	0.5	0.116731	88.333333
69	Cosine-w2v	evictions.csv	agency_spending.csv	96.0	11.0	1	0.0	0.3	0.089531	88.541667
70	Cosine-w2v	evictions.csv	capital_projects1.csv	60.0	16.0	3	0.0	0.2	0.053791	73.333333
71	Cosine-w2v	recent_contract_awards.csv	film_permits.csv	144.0	97.0	8	0.0	0.1	0.135726	32.638889
72	Cosine-w2v	recent_contract_awards.csv	housing.csv	120.0	47.0	4	0.0	0.2	0.119891	60.833333
73	Cosine-w2v	recent_contract_awards.csv	agency_spending.csv	96.0	5.0	2	0.0	0.6	0.093921	94.791667
74	Cosine-w2v	recent_contract_awards.csv	capital_projects1.csv	60.0	14.0	6	0.0	0.3	0.060715	76.666667
75	Cosine-w2v	housing.csv	film_permits.csv	120.0	117.0	3	0.0	0.0	0.114649	2.500000
76	Cosine-w2v	housing.csv	capital_projects2.csv	40.0	2.0	2	0.0	0.8	0.036864	95.000000

FIGURE A.1: Threshold Exploration Cosine-w2v NY Data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time
84	Cosine-w2v	clients_registered_lob_firms.csv	payments_clients_lob_firms.csv	52.0	11.0	4	0.0	0.4	0.047930
85	Cosine-w2v	clients_registered_lob_firms.csv	city_proj_agencies_lob_firms.csv	78.0	15.0	5	0.0	0.4	0.073111
86	Cosine-w2v	payments_clients_lob_firms.csv	city_proj_agencies_lob_firms.csv	24.0	4.0	4	0.0	0.8	0.023308
87	Cosine-w2v	city_campaign_contribs.csv	clients_registered_lob_firms.csv	182.0	147.0	16	0.0	0.1	0.171329
88	Cosine-w2v	city_campaign_contribs.csv	payments_clients_lob_firms.csv	56.0	50.0	1	0.0	0.1	0.055246
89	Cosine-w2v	city_campaign_contribs.csv	city_proj_agencies_lob_firms.csv	84.0	68.0	1	0.0	0.1	0.077938
90	Cosine-w2v	asset_inventory.csv	clients_registered_lob_firms.csv	208.0	7.0	6	0.0	0.5	0.201013
91	Cosine-w2v	asset_inventory.csv	city_campaign_contribs.csv	224.0	216.0	9	0.0	0.0	0.209214
92	Cosine-w2v	election_list.csv	city_campaign_contribs.csv	98.0	19.0	2	0.0	0.3	0.091289
93	Cosine-w2v	election_list.csv	clients_registered_lob_firms.csv	91.0	4.0	2	0.0	0.4	0.082829
94	Cosine-w2v	city_reg_lobbyists.csv	city_proj_agencies_lob_firms.csv	84.0	26.0	5	0.0	0.4	0.077099
95	Cosine-w2v	city_reg_lobbyists.csv	payments_clients_lob_firms.csv	56.0	22.0	5	0.0	0.4	0.054285
96	Cosine-w2v	city_reg_lobbyists.csv	clients_registered_lob_firms.csv	182.0	32.0	19	0.0	0.5	0.184205
97	Cosine-w2v	city_reg_lobbyists.csv	city_campaign_contribs.csv	196.0	99.0	19	0.0	0.2	0.180033
98	Cosine-w2v	city_reg_lobbyists.csv	asset_inventory.csv	224.0	211.0	11	0.0	0.0	0.215383
99	Cosine-w2v	clients_registered_lob_firms.csv	asset_inventory.csv	208.0	202.0	4	0.0	0.0	0.211326
100	Cosine-w2v	payments_clients_lob_firms.csv	asset_inventory.csv	64.0	57.0	4	0.0	0.0	0.060275
101	Cosine-w2v	city_proj_agencies_lob_firms.csv	asset_inventory.csv	96.0	88.0	4	0.0	0.0	0.089690
102	Cosine-w2v	clients_registered_lob_firms.csv	city_campaign_contribs.csv	182.0	147.0	1	0.0	0.1	0.204438
103	Cosine-w2v	payments_clients_lob_firms.csv	city_campaign_contribs.csv	56.0	50.0	1	0.0	0.1	0.055810

FIGURE A.2: Threshold Exploration Cosine-w2v LA Data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time
28	Jaro-Winkler	civil_list.csv	evictions.csv	48.0	30.0	2	0.0	0.5	0.032458
29	Jaro-Winkler	civil_list.csv	city_record_online.csv	60.0	34.0	2	0.0	0.5	0.041664
30	Jaro-Winkler	civil_list.csv	agency_spending.csv	32.0	8.0	2	0.0	0.6	0.021499
31	Jaro-Winkler	city_record_online.csv	film_permits.csv	180.0	169.0	11	0.0	0.4	0.124953
32	Jaro-Winkler	city_record_online.csv	recent_contract_awards.csv	180.0	118.0	13	0.0	0.5	0.136208
33	Jaro-Winkler	city_record_online.csv	agency_spending.csv	120.0	115.0	3	0.0	0.4	0.082406
34	Jaro-Winkler	city_record_online.csv	capital_projects1.csv	75.0	58.0	9	0.0	0.5	0.054919
35	Jaro-Winkler	city_record_online.csv	evictions.csv	180.0	171.0	5	0.0	0.4	0.122813
36	Jaro-Winkler	city_record_online.csv	housing.csv	150.0	138.0	4	0.0	0.4	0.118558
37	Jaro-Winkler	evictions.csv	film_permits.csv	144.0	82.0	5	0.0	0.5	0.102787
38	Jaro-Winkler	evictions.csv	recent_contract_awards.csv	144.0	96.0	4	0.0	0.5	0.099774
39	Jaro-Winkler	evictions.csv	city_record_online.csv	180.0	38.0	2	0.0	0.6	0.130047
40	Jaro-Winkler	evictions.csv	housing.csv	120.0	65.0	3	0.0	0.5	0.086487
41	Jaro-Winkler	evictions.csv	agency_spending.csv	96.0	12.0	1	0.0	0.6	0.072750
42	Jaro-Winkler	evictions.csv	capital_projects1.csv	60.0	47.0	3	0.0	0.5	0.045458
43	Jaro-Winkler	recent_contract_awards.csv	film_permits.csv	144.0	137.0	8	0.0	0.4	0.104297
44	Jaro-Winkler	recent_contract_awards.csv	housing.csv	120.0	113.0	4	0.0	0.4	0.085606
45	Jaro-Winkler	recent_contract_awards.csv	agency_spending.csv	96.0	92.0	2	0.0	0.4	0.071323
46	Jaro-Winkler	recent_contract_awards.csv	capital_projects1.csv	60.0	47.0	6	0.0	0.5	0.042874
47	Jaro-Winkler	housing.csv	film_permits.csv	120.0	55.0	3	0.0	0.5	0.088429
48	Jaro-Winkler	housing.csv	capital_projects2.csv	40.0	3.0	2	0.0	0.9	0.027545
49	Jaro-Winkler	evictions.csv	capital_projects2.csv	48.0	1.0	1	0.0	0.7	0.032981
50	Jaro-Winkler	agency_spending.csv	city_record_online.csv	120.0	1.0	1	0.0	0.8	0.086498
51	Jaro-Winkler	agency_spending.csv	recent_contract_awards.csv	96.0	1.0	1	0.0	0.8	0.067215
52	Jaro-Winkler	agency_spending.csv	capital_projects1.csv	40.0	33.0	3	0.0	0.5	0.029380
53	Jaro-Winkler	agency_spending.csv	film_permits.csv	96.0	87.0	3	0.0	0.4	0.071297
54	Jaro-Winkler	capital_projects1.csv	film_permits.csv	60.0	59.0	9	0.0	0.4	0.041549
55	Jaro-Winkler	capital_projects2.csv	film_permits.csv	48.0	2.0	1	0.0	0.9	0.032353

FIGURE A.3: Threshold Exploration Jaro-Winkler NY Data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time
42	Jaro-Winkler	clients_registered_lob_firms.csv	payments_clients_lob_firms.csv	52.0	11.0	4	0.0	0.7	0.039166
43	Jaro-Winkler	clients_registered_lob_firms.csv	city_proj_agencies_lob_firms.csv	78.0	12.0	5	0.0	0.7	0.058134
44	Jaro-Winkler	payments_clients_lob_firms.csv	city_proj_agencies_lob_firms.csv	24.0	4.0	4	0.0	0.9	0.016586
45	Jaro-Winkler	city_campaign_contribs.csv	clients_registered_lob_firms.csv	182.0	139.0	16	0.0	0.5	0.143502
46	Jaro-Winkler	city_campaign_contribs.csv	payments_clients_lob_firms.csv	56.0	44.0	1	0.0	0.5	0.040557
47	Jaro-Winkler	city_campaign_contribs.csv	city_proj_agencies_lob_firms.csv	84.0	64.0	1	0.0	0.5	0.058871
48	Jaro-Winkler	asset_inventory.csv	clients_registered_lob_firms.csv	208.0	100.0	6	0.0	0.5	0.146699
49	Jaro-Winkler	asset_inventory.csv	city_campaign_contribs.csv	224.0	210.0	9	0.0	0.4	0.169832
50	Jaro-Winkler	election_list.csv	city_campaign_contribs.csv	98.0	76.0	2	0.0	0.5	0.073870
51	Jaro-Winkler	election_list.csv	clients_registered_lob_firms.csv	91.0	22.0	2	0.0	0.6	0.064622
52	Jaro-Winkler	city_reg_lobbyists.csv	city_proj_agencies_lob_firms.csv	84.0	34.0	5	0.0	0.6	0.064603
53	Jaro-Winkler	city_reg_lobbyists.csv	payments_clients_lob_firms.csv	56.0	32.0	5	0.0	0.6	0.039394
54	Jaro-Winkler	city_reg_lobbyists.csv	clients_registered_lob_firms.csv	182.0	150.0	19	0.0	0.5	0.131571
55	Jaro-Winkler	city_reg_lobbyists.csv	city_campaign_contribs.csv	196.0	188.0	19	0.0	0.4	0.142908
56	Jaro-Winkler	city_reg_lobbyists.csv	asset_inventory.csv	224.0	192.0	11	0.0	0.4	0.155437
57	Jaro-Winkler	clients_registered_lob_firms.csv	asset_inventory.csv	208.0	206.0	4	0.0	0.0	0.168435
58	Jaro-Winkler	payments_clients_lob_firms.csv	asset_inventory.csv	64.0	62.0	4	0.0	0.0	0.044746
59	Jaro-Winkler	city_proj_agencies_lob_firms.csv	asset_inventory.csv	96.0	92.0	4	0.0	0.0	0.074613
60	Jaro-Winkler	clients_registered_lob_firms.csv	city_campaign_contribs.csv	182.0	139.0	1	0.0	0.5	0.128724
61	Jaro-Winkler	payments_clients_lob_firms.csv	city_campaign_contribs.csv	56.0	44.0	1	0.0	0.5	0.040460
62	Jaro-Winkler	city_proj_agencies_lob_firms.csv	city_campaign_contribs.csv	84.0	64.0	1	0.0	0.5	0.058197
63	Jaro-Winkler	city_reg_lobbyists.csv	election_list.csv	98.0	53.0	3	0.0	0.5	0.069596
64	Jaro-Winkler	clients_registered_lob_firms.csv	election_list.csv	91.0	22.0	1	0.0	0.6	0.062298
65	Jaro-Winkler	positions.csv	asset_inventory.csv	96.0	1.0	2	0.0	0.9	0.064746
66	Jaro-Winkler	open_budget.csv	clients_registered_lob_firms.csv	104.0	58.0	1	0.0	0.5	0.073771
67	Jaro-Winkler	open_budget.csv	payments_clients_lob_firms.csv	32.0	7.0	1	0.0	0.6	0.022078
68	Jaro-Winkler	open_budget.csv	city_proj_agencies_lob_firms.csv	48.0	7.0	1	0.0	0.6	0.034794
69	Jaro-Winkler	open_budget.csv	city_reg_lobbyists.csv	112.0	58.0	1	0.0	0.5	0.078888
70	Jaro-Winkler	positions.csv	open_budget.csv	48.0	4.0	1	0.0	0.9	0.034284

FIGURE A.4: Threshold Exploration Jaro-Winkler LA Data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time
0	Levenshtein	civil_list.csv	evictions.csv	48.0	21.0	2	0.0	13.0	0.032002
1	Levenshtein	civil_list.csv	city_record_online.csv	60.0	15.0	2	0.0	9.0	0.039263
2	Levenshtein	civil_list.csv	agency_spending.csv	32.0	21.0	2	0.0	15.0	0.030433
3	Levenshtein	city_record_online.csv	film_permits.csv	180.0	138.0	11	0.0	14.0	0.122256
4	Levenshtein	city_record_online.csv	recent_contract_awards.csv	180.0	109.0	13	0.0	13.0	0.121960
5	Levenshtein	city_record_online.csv	agency_spending.csv	120.0	43.0	3	0.0	12.0	0.080665
6	Levenshtein	city_record_online.csv	capital_projects1.csv	75.0	33.0	9	3.0	11.0	0.051595
7	Levenshtein	city_record_online.csv	evictions.csv	180.0	79.0	5	1.0	14.0	0.125091
8	Levenshtein	city_record_online.csv	housing.csv	150.0	71.0	4	0.0	13.0	0.100363
9	Levenshtein	evictions.csv	film_permits.csv	144.0	54.0	5	0.0	13.0	0.103450
10	Levenshtein	evictions.csv	recent_contract_awards.csv	144.0	69.0	4	0.0	15.0	0.098795
11	Levenshtein	evictions.csv	city_record_online.csv	180.0	52.0	2	0.0	12.0	0.121857
12	Levenshtein	evictions.csv	housing.csv	120.0	20.0	3	0.0	11.0	0.082726
13	Levenshtein	evictions.csv	agency_spending.csv	96.0	7.0	1	0.0	10.0	0.065259
14	Levenshtein	evictions.csv	capital_projects1.csv	60.0	13.0	3	1.0	12.0	0.041089
15	Levenshtein	recent_contract_awards.csv	film_permits.csv	144.0	104.0	8	0.0	14.0	0.097086
16	Levenshtein	recent_contract_awards.csv	housing.csv	120.0	53.0	4	0.0	13.0	0.084157
17	Levenshtein	recent_contract_awards.csv	agency_spending.csv	96.0	34.0	2	0.0	12.0	0.067982
18	Levenshtein	recent_contract_awards.csv	capital_projects1.csv	60.0	27.0	6	2.0	11.0	0.042058
19	Levenshtein	housing.csv	film_permits.csv	120.0	45.0	3	0.0	11.0	0.080623
20	Levenshtein	housing.csv	capital_projects2.csv	40.0	2.0	2	0.0	5.0	0.028092
21	Levenshtein	evictions.csv	capital_projects2.csv	48.0	1.0	1	0.0	5.0	0.033837
22	Levenshtein	agency_spending.csv	city_record_online.csv	120.0	1.0	1	0.0	5.0	0.080515
23	Levenshtein	agency_spending.csv	recent_contract_awards.csv	96.0	1.0	1	0.0	5.0	0.065212
24	Levenshtein	agency_spending.csv	capital_projects1.csv	40.0	22.0	3	1.0	15.0	0.026945
25	Levenshtein	agency_spending.csv	film_permits.csv	96.0	57.0	3	0.0	15.0	0.062598
26	Levenshtein	capital_projects1.csv	film_permits.csv	60.0	48.0	9	2.0	15.0	0.039299
27	Levenshtein	capital_projects2.csv	film_permits.csv	48.0	2.0	1	0.0	5.0	0.031962

FIGURE A.5: Threshold Exploration Levenshtein NY Data

	Method	Dataset_1	Dataset_2	Total_Pairs	Selected_Pairs	Actual_Pairs	Missing	Threshold	Time
0	Levenshtein	clients_registered_lob_firms.csv	payments_clients_lob_firms.csv	52.0	29.0	4	0.0	15.0	0.034616
1	Levenshtein	clients_registered_lob_firms.csv	city_proj_agencies_lob_firms.csv	78.0	39.0	5	0.0	15.0	0.051684
2	Levenshtein	payments_clients_lob_firms.csv	city_proj_agencies_lob_firms.csv	24.0	4.0	4	0.0	5.0	0.016580
3	Levenshtein	city_campaign_contribs.csv	clients_registered_lob_firms.csv	182.0	109.0	16	5.0	15.0	0.129846
4	Levenshtein	city_campaign_contribs.csv	payments_clients_lob_firms.csv	56.0	6.0	1	0.0	11.0	0.039586
5	Levenshtein	city_campaign_contribs.csv	city_proj_agencies_lob_firms.csv	84.0	10.0	1	0.0	11.0	0.058974
6	Levenshtein	asset_inventory.csv	clients_registered_lob_firms.csv	208.0	122.0	6	4.0	14.0	0.143874
7	Levenshtein	asset_inventory.csv	city_campaign_contribs.csv	224.0	159.0	9	4.0	15.0	0.153774
8	Levenshtein	election_list.csv	city_campaign_contribs.csv	98.0	32.0	2	0.0	13.0	0.068574
9	Levenshtein	election_list.csv	clients_registered_lob_firms.csv	91.0	22.0	2	0.0	12.0	0.065320
10	Levenshtein	city_reg_lobbyists.csv	city_proj_agencies_lob_firms.csv	84.0	10.0	5	1.0	11.0	0.056621
11	Levenshtein	city_reg_lobbyists.csv	payments_clients_lob_firms.csv	56.0	10.0	5	1.0	11.0	0.039674
12	Levenshtein	city_reg_lobbyists.csv	clients_registered_lob_firms.csv	182.0	97.0	19	2.0	15.0	0.125567
13	Levenshtein	city_reg_lobbyists.csv	city_campaign_contribs.csv	196.0	118.0	19	5.0	15.0	0.132296
14	Levenshtein	city_reg_lobbyists.csv	asset_inventory.csv	224.0	119.0	11	8.0	14.0	0.163753
15	Levenshtein	clients_registered_lob_firms.csv	asset_inventory.csv	208.0	0.0	4	4.0	5.0	0.144211
16	Levenshtein	payments_clients_lob_firms.csv	asset_inventory.csv	64.0	0.0	4	4.0	5.0	0.044901
17	Levenshtein	city_proj_agencies_lob_firms.csv	asset_inventory.csv	96.0	0.0	4	4.0	5.0	0.065548
18	Levenshtein	clients_registered_lob_firms.csv	city_campaign_contribs.csv	182.0	0.0	1	1.0	5.0	0.125036
19	Levenshtein	payments_clients_lob_firms.csv	city_campaign_contribs.csv	56.0	0.0	1	1.0	5.0	0.040162
20	Levenshtein	city_proj_agencies_lob_firms.csv	city_campaign_contribs.csv	84.0	0.0	1	1.0	5.0	0.056301
21	Levenshtein	city_reg_lobbyists.csv	election_list.csv	98.0	27.0	3	1.0	13.0	0.069081
22	Levenshtein	clients_registered_lob_firms.csv	election_list.csv	91.0	0.0	1	1.0	5.0	0.064917
23	Levenshtein	positions.csv	asset_inventory.csv	96.0	5.0	2	0.0	6.0	0.066677
24	Levenshtein	open_budget.csv	clients_registered_lob_firms.csv	104.0	7.0	1	0.0	10.0	0.072141
25	Levenshtein	open_budget.csv	payments_clients_lob_firms.csv	32.0	1.0	1	0.0	7.0	0.023386
26	Levenshtein	open_budget.csv	city_proj_agencies_lob_firms.csv	48.0	1.0	1	0.0	7.0	0.032854
27	Levenshtein	open_budget.csv	city_reg_lobbyists.csv	112.0	2.0	1	0.0	10.0	0.085389
28	Levenshtein	positions.csv	open_budget.csv	48.0	6.0	1	0.0	5.0	0.036540
29	Levenshtein	open_budget.csv	positions.csv	48.0	6.0	3	0.0	5.0	0.034160
30	Levenshtein	open_budget.csv	asset_inventory.csv	128.0	23.0	2	0.0	10.0	0.089971

FIGURE A.6: Threshold Exploration Levenshtein LA Data

Bibliography

A. A. Alwan, A. Nordin, M. Alzeber, and A. Z. Abualkishik. A survey of schema matching research using database schemas and instances. *International Journal of Advanced Computer Science and Applications*, 8(10), 2017.

M. B. Amin, W. A. Khan, S. Hussain, D.-M. Bui, O. Banos, B. H. Kang, and S. Lee. Evaluating large-scale biomedical ontology matching over parallel platforms. *Iete Technical Review*, 33(4):415–427, 2016.

Y. Amsterdamer, S. B. Davidson, A. Kukliansky, T. Milo, S. Novgorodov, and A. Somech. Managing general and individual knowledge in crowd mining applications. In *CIDR*, 2015.

I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann, et al. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in neuroanatomy*, page 142, 2015.

D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, 2005.

P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *ACM Sigmod Record*, 33(4):38–43, 2004.

P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.

R. M. Borromeo and M. Toyama. Automatic vs. crowdsourced sentiment analysis. In *Proceedings of the 19th International Database Engineering & Applications Symposium*, pages 90–95, 2015.

S. Castano and V. De Antonellis. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.

F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Computing Surveys (CSUR)*, 51(1):1–40, 2018.

G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *The VLDB Journal*, 22(5): 665–687, 2013.

H.-H. Do and E. Rahm. Coma—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 610–621. Elsevier, 2002.

A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 509–520, 2001.

F. Duchateau and Z. Bellahsene. Designing a benchmark for the assessment of schema matching tools. *Open Journal of Databases*, 1(1):3–25, 2014.

F. Duchateau, Z. Bellahsene, and E. Hunt. Xbenchmatch: a benchmark for xml schema matching tools. In *The VLDB Journal*, volume 1, pages 1318–1321. Springer Verlag, 2007.

M. Ehrig and S. Staab. Qom-quick ontology mapping. In *ISWC*, pages 683–697. Springer, 2004.

J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *2014 IEEE 30th International Conference on Data Engineering*, pages 976–987. IEEE, 2014.

A. Finnerty, P. Kucherbaev, S. Tranquillini, and G. Convertino. Keep it simple: Reward and task design in crowdsourcing. In *Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*, pages 1–4, 2013.

A. Gross, M. Hartung, T. Kirsten, and E. Rahm. On matching large life science ontologies in parallel. In *Data Integration in the Life Sciences: 7th International Conference, DILS 2010, Gothenburg, Sweden, August 25-27, 2010. Proceedings 7*, pages 35–49. Springer, 2010.

D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *arXiv preprint arXiv:1509.05969*, 2015.

N. Q. V. Hung, N. T. Tam, Z. Miklós, and K. Aberer. On leveraging crowdsourcing techniques for schema matching networks. In *International Conference on Database Systems for Advanced Applications*, pages 139–154. Springer, 2013.

C. Koutras, K. Psarakis, G. Siachamis, A. Ionescu, M. Fragkoulis, A. Bonifati, and A. Katsifodimos. Valentine in action: matching tabular data at scale. *Proceedings of the VLDB Endowment*, 14(12):2871–2874, 2021.

C. Koutras, R. Hai, K. Psarakis, M. Fragkoulis, and A. Katsifodimos. Sima: Effective and efficient data silo federation using graph neural networks. *arXiv preprint arXiv:2206.12733*, 2022.

G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng. Crowdsourced data management: Overview and challenges. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1711–1716, 2017.

J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *vldb*, volume 1, pages 49–58, 2001.

J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *21st International Conference on Data Engineering (ICDE'05)*, pages 57–68. IEEE, 2005.

S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th international conference on data engineering*, pages 117–128. IEEE, 2002.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *Proceedings of the VLDB Endowment*, 11(7):813–825, 2018.

F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

E. Peukert, H. Berthold, and E. Rahm. Rewrite techniques for performance optimization of schema matching processes. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 453–464, 2010.

E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10:334–350, 2001.

E. Rahm and E. Peukert. Large-scale schema matching., 2019.

P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. In *Journal on data semantics IV*, pages 146–171. Springer, 2005.

L. Singh. Clustering text: A comparison between available text vectorization techniques. In *Soft Computing and Signal Processing: Proceedings of 3rd ICSCSP 2020, Volume 2*, pages 21–27. Springer, 2022.

S. Vijayarani, M. J. Ilamathi, M. Nithya, et al. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *arXiv preprint arXiv:1208.1927*, 2012.

C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proceedings of the VLDB Endowment*, 6(9):757–768, 2013.

C. J. Zhang, Z. Zhao, L. Chen, H. V. Jagadish, and C. C. Cao. Crowdmatcher: crowd-assisted schema matching. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 721–724, 2014.

H. Zheng, D. Li, and W. Hou. Task design, motivation, and participation in crowdsourcing contests. *International Journal of Electronic Commerce*, 15(4):57–88, 2011.

E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 847–864, 2019.