Table Union Search with Preferences

by

Hamed Mirzaei

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

We study the problem of Table Union Search (TUS) in the presence of preferences. Two tables are unionable if their column values are drawn from the same domains. This notion of unionability is too coarse to be effective in down-stream tasks. The result of a table search for unionability is often less relevant to the needs of users, and selecting top few is subjective and depends on the follow-up operations.

This thesis introduces preferences for table unionability, as a way to reduce the search space and focus on rows and columns that are important for the follow-up operations. But, adding preferences introduces a few challenges to the process. Firstly, one may need extra information such as the relationship between webtables which TUS does not consider. Secondly, there is usually additional overload, which can be costly when searching a large set of webtables. We study a few approaches to address these challenges.

We evaluate the efficiency and effectiveness of preferences on three down-stream tasks, showing that adding preferences significantly improves the performance of these tasks.

Preface

We intend to publish this thesis as a full paper. We have the draft version of the paper ready and it is under our review right now.

I Wonder... What If? Let's Try!

– Sesame Street TV Series

Acknowledgements

To begin with, I would like to convey my deepest and most sincere appreciation to Professor Davood Rafiei, my supervisor, without whom this project would not have been possible. Prof. Rafiei generously devoted his time and energy to assist me in my research, providing invaluable guidance and sharing brilliant ideas. His encouragement to delve more deeply and critically into possible solutions, along with his insightful and constructive feedback, proved to be instrumental in this endeavor. He was remarkably patient and understanding, especially during times when I required assistance. It was a great honor for me to have such a fantastic mentor.

I am also grateful to Professor Zachary Friggstad, who was always ready to lend a helping hand whenever I encountered difficulties with an algorithmic issue.

I would like to express my heartfelt appreciation to my dear friend, Farzaneh Sepehr, who has always been a supportive presence in my life druing this program.

Additionally, I want to express my gratitude to my parents, Ali Yavar and Mahieh, for their unwavering love and support, as well as to my siblings, Nahid, Negar, Rasoul, Ali, Sajjad, Farshad, Mehrdad, and Reza, for their constant presence in my life.

Finally, I wish to thank my friends for contributing to the enjoyment and interest of this program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The web contains a vast corpus of relational tables. Those tables are a great source of structural information and can be a valuable resource in many use cases and applications (e.g. table augmentation [22], [86], [89], knowledge base population [64], [90] and question answering [53], [67]). *Table Union Search (TUS)* is an operation where one wants to find webtables that can be unioned with a query table. The notion of unionability in relational databases is defined as follows: *Two tables are considered unionable if their column values are drawn from the same domains.* However, in the context of the web, the domains of columns are not known or fixed, unlike relational databases where the domains are well-defined. For the same reason, existing approaches for TUS often rely on value overlap to find columns with the same domains. Following this idea, we may have columns with different levels of overlap with query columns, leading to webtables with varying degrees of unionability. Also, webtables may have different subsets of columns with value overlap with query columns. Using all this information, TUS ranks webtables according to their unionability, with the most unionable table to the query ranked the highest. However, using this generic notion of unionability in down-stream tasks or with other operations can be challenging.

First, the value overlap is not always a good measure for unionability. Consider, for instance, two columns containing country names, one for Asia and one for America. There is no overlap between the two columns, but they are both drawn from the same domains, countries. Existing approaches use other

unionability measures such as the cosine similarity between word embedding representations of the two columns' values or the overlap between the corresponding entity types of columns' values extracted from a knowledge graph [52]. Second, the output size of TUS is usually large, leaving the user with many unionable webtables and making it difficult to extract the information they need. Some works try to address this issue by focusing on returning a top-k list of most unionable webtables in response to a given query table [52] which brings us to the next challenge. As the third challenge, no matter what the user prefers to see in the results and what follow-up operation will be applied on the query table, TUS returns the same list of webtables in response to a specific query table. Ranking the webtables or selecting the top few is a subjective task, and depending on the user's needs, it may be better to return a different ranked list (i.e. it could be the same list, but in a different order). Finally, TUS is also limited in its ability to identify and exploit complex relationships between webtables and columns. Different webtables and different columns of the same table are often regarded as independent entities, and this results in not only missing valuable information but also returning near-duplicate webtables that add no new information to the query table.

In this thesis, we extend TUS with preferences to address the above-mentioned issues. Each preference addresses some of the aforementioned problems and helps the follow-up operations. We also study the problem of efficiently evaluating TUS under preferences.

1.1 Motivating Example

Consider the query table Q and webtables $C_1 - C_3$, all containing information about countries crawled from the web, as shown in Table 1.1. The query table lists a few countries with their populations while missing the population of Iran. C_1 is a duplicate of Q with the exception that it has no missing value. C_2 has one of the records in Q and one new record. It also has a new column called 'area'. C_3 contains one of the records of Q and two other new records. All these tables can be considered union compatible to Q, since they all offer

Query Table Q	
country	**population**
Canada	38.25
Iran	NULL
USA	331.9

Webtable C1	
country	**population**
Canada	38.25
Iran	85.03
USA	331.9

Webtable C2		
country	**population**	**area**
Canada	38.25	9.985
Germany	83.13	0.357

Webtable C3	
country	**population**
Canada	38.25
France	67.5
India	1393

Table 1.1: Motivating example: query table Q and candidate webtables $C_1 - C_3$

columns with the same name and domain as those of Q.

Consider a user who wants to use Q as the train data for a machine learning model that is highly sensitive to missing values. The user decides to use TUS to find the missing value in Q since the table has missing information. TUS would fulfill the user's request if it favors C_1 over other tables since it contains the missing value of Q, i.e. the population of Iran. Consider another user who wants to train a machine learning model using records of Q, but due to the small size of the training data, the model cannot capture the underlying trend and suffers from underfitting. To get unionable webtables with extra records for Q, the user executes TUS over Q. Returning C_3 offers more new records for Q in this case. In a completely different scenario, the user may need more features for Q with the goal of training a more accurate machine learning model. This time, extending Q horizontally and adding more columns would be the reason behind executing TUS. Consequently, C_2 by offering another feature, 'area' column, is the best choice to return to the user.

1.2 Problem Statement

Without knowing the end user's goal or the next follow-up operation, the TUS operation must return all tables that are unionable with the query table. However, returning a large output size makes it hard for the user to process all the returned unionable webtables and extract the desired information. An

approach that is adopted is to return only the top most unionable webtables. We argue that, returning the same list of top most unionable webtables in response to different follow-up operations is not beneficial for the user. More precisely, there is no one-size-fits-all definition for TUS since it is a subjective operation. The purpose of this research is to introduce preferences to TUS in order to efficiently and effectively return the top unionable webtables with the highest benefit in response to different follow-up operations.

1.3 Challenges

Preferences have been studied in different domains such as databases [71] and web search [62] to prevent returning either empty or excessively large results to user queries. They are soft constraints that are provided by the user and describe how the results are computed or the properties that the results should possess. In this work we investigate if the notion of preferences is meaningful for TUS operation or not. In particular, we explore those preferences that have the potential to add benefits to TUS operation. As the ultimate goal is to meet the expectations of a user who wants to utilize the output of TUS for different follow-up operations, we delve into the details of how different preferences can be added to TUS. In this process, we identify some useful preferences and explore the challenges of adding them to TUS.

Although each preference has its unique set of challenges, there are some general challenges that need to be explored before adding any preference to TUS. The first challenge is that the existing treatment of preferences in the literature may not be directly applicable to webtables. One may need to transform webtables to a specific format or structure [42], [73]. This can be challenging if the transformation operation needs to be efficient without losing any valuable information in order to be applicable in real-time environments. For instance, *Skyline* [17], [36] usually works with data points represented as vectors with numeric elements. There is a challenge in representing each webtable with various levels of unionability with the query table in a single vector, or multiple vectors, without losing information.

The second challenge is that preferences may need to materialize large intermediate results, e.g. by computing the relationship between webtables and columns and maintaining those relationships in an efficient data structure such as B^+-tree [76]. Although these intermediate results can help later in efficiently ranking the results, they add another layer of computation which may have a negative impact on the efficiency of the whole approach. Additionally, some preferences require searching over a large range of webtables and columns, even those with low unionability. An instance of this is when the preference is the diversity of the webtables that are returned[12]. Since high unionable tables are likely to be from the same domain as the query table, a low unionability may be preferred. This can make the use of preferences less scalable and less efficient, which may not be desirable for a real-time environment.

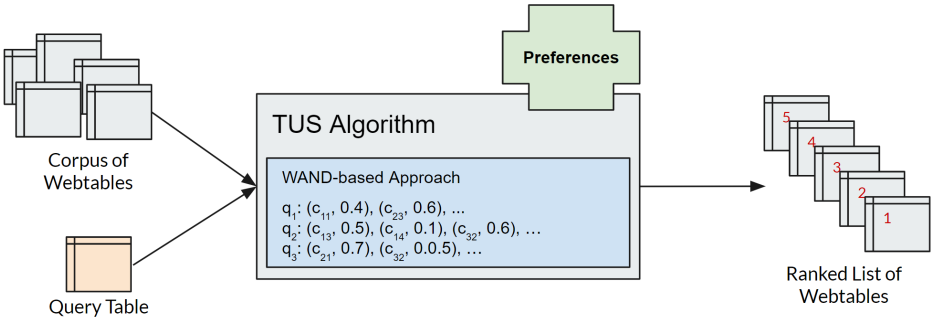
The third challenge relates to the information that may not be preserved under unionability about query table or candidate webtables. For example, TUS does not capture the relationships between different columns of a table or the relationships between different candidate webtables, and it assumes they are independent entities. This makes preferences that rely on such dependencies inapplicable [51], [52]. An example preference might search for candidate webtables that have the same combination of values over a subset of query columns. As a result, the user is guaranteed that the returned webtables are both unionable with the query table and have common records. When the columns are considered independent as they are in TUS, it is not possible to apply this preference.

In this thesis, we investigate these challenges and propose a few well-defined preferences to TUS.

1.4 Overview of Our Approach

The approach we present in our research aims to rank a corpus of candidate webtables based on their unionability with a query table. This process involves taking both the query table and candidate webtables as input, and then using the preferences to determine the best matches. Preferences can be incorpo-

Figure 1.1: Overview of the approach



rated in various ways, including as a pre-processing step, a post-processing step, or interleaved with the table union search operation. Figure 1.1 provides an overview of our approach, which outlines the different steps involved in the process. The main advantage of our approach is that it allows users to quickly find the most relevant webtables that match their query, saving them time and effort. Additionally, by incorporating preferences, users can tailor the search to their specific needs, ensuring that the results are both relevant and useful.

In this thesis, we first present the notion of unionability, starting with column unionability and building upon it the table unionability. To address the limitations of TUS, as discussed above, we propose some major preferences. For each preference, we investigate whether specific data mappings are needed or additional information needs to be maintained for these preferences to be included in TUS. Finally, to efficiently incorporate preferences, we employ multiple strategies including i) pre-computing some information such as the relationships between webtables in the dataset (Section 3.4.2), ii) exploiting the WAND [9] approach and working with posting lists of query columns in order to prune webtables early in the process (Section 3.2), iii) developing greedy algorithms to optimize the search (Section 3.4), and iv) partitioning the input dataset into smaller chunks and processing them in parallel (Section 3.4.1).

1.5 Research Contributions

Our aim is to incorporate preferences into the TUS operation and to return top unionable webtables for various follow-up operations. The contributions of this thesis are as follows:

- We introduce four major preferences to the TUS operation: skyline, novelty, diversity and dependent set. The first three have been introduced in the literature but not specifically over the TUS operation. The last one, dependent set, is a novel preference proposed by this work.
- We develop two different algorithms for evaluating TUS with our proposed preferences. One approach uses the past work on TUS[52] but utilizes the *WAND* approach [9] to prune many webtables without fully examining them, and the other approach adds filters such as 'candidate webtable should have an alignment over a key of the query table' to the first approach.
- We introduce two benchmark datasets for unionable webtables, constructed from *Web Data Commons - English-Language Relational Web Tables 2015 (WDC)* dataset [48] and *WikiTables* dataset [5]. This is done by performing selection and projection operations on webtables followed by randomly masking some values. The candidate webtables are generated so that they also cover rows and columns not selected for the query webtables. This guarantees the existence of new rows and columns for the query webtables in the datasets, which are important for evaluating TUS and preferences on webtables.
- We propose efficient approaches for evaluating each preference. Our experimental evaluation shows that using preferences leads to higher quality results compared to the two implementations of TUS over different down-stream tasks.

1.6 Thesis Organization

In Chapter 2, we discuss the related work in the areas of table union search and preference queries. In Chapter 3, we introduce TUS, discuss the detail of an evaluation strategy, introduce our preferences, and discuss how they can be efficiently evaluated. In Chapter 4, we present our evaluation on two widely used datasets and over three down-stream tasks. Finally, in Chapter 5, we conclude the thesis and discuss the future directions.

Chapter 2

Related Works

As this research resides in the intersection area between table search and preferences, our coverage of the related works is broken down into these two topics. Table search works focus on finding unionable tables for a given query and preferences help with returning customized results based on user's interests.

2.1 Table Search

Table search belongs to a wider area of research, under data discovery and data integration, which includes relevant topics such as table stitching, table augmentation, table extraction, etc. The search for unionable tables may be done for a table (table-based search) or a list of keywords (keyword-based search) as the search query. In both cases, the result is a ranked list of tables based on their similarity/relevance/unionability with the search query. Table search is an important task of data discovery and may be used in other tasks such as table augmentation [22], [49], [86], [88], knowledge base augmentation [6], [64] and question answering [58], [72].

2.1.1 Keyword-Based Search

Works in this area can be broken into two categories of *document-based* and *feature-based* approaches. The document-based methods represent each candidate table as a document and exploit information retrieval methods to find the most relevant tables. For example, Pimplikar et al. [60] uses information such as the content and the column headers of a candidate table as part of its

corresponding document representation.

However, most of the works are feature-based approaches which represent both the search query and candidate tables as sets of features and use traditional approaches like machine learning models to calculate their unionability. Cafarella et al. [11] introduces a keyword-based table search on top of an existing web search engine to extract the top-k tables from the returned webpages. Their ranking method uses table-structure-aware features such as the number of query terms in table's header and body and query-independent table coherence score. Later, they extend their method with a stronger re-ranking mechanism, *SCPRank*, introduced within *OCTOPUS* [10]. *SCPRank* calculates the correlation between each query term and each cell of the table using a symmetric conditional probability.

Pimplikar et al. [60] designs a structured search engine called *WWT* which takes as input multiple sets of column keywords each describing one column of the desired candidate tables. They first use all the column keywords to get the first group of candidate tables with the best hits of keywords in their content, context or column headers. Then, candidate tables with a value overlap with the first group will be added to the candidate list as potential candidates. Finally, they make a single table out of all the returned candidate tables by merging all their relevant columns and rows.

Zhang et al. [87] represents search query and candidate tables in various semantic spaces such as continuous embedding dense vectors and discrete bag-of-concepts sparse vectors. They use features such as the page title, the table caption and the table body for candidate tables and the number of terms and the IDF score of each term for the search query. Then, they exploit different similarity measures like *early fusion* and *late fusion* to calculate the similarity of search query and candidate tables. Finally, they use different combinations of semantic spaces and similarity measures as features of a supervised learning model and show a significant improvements over the state-of-the-art baselines.

Deng et al. [25] employs the skip-gram model of *Word2Vec* and proposes a method for generating neural embeddings for tables that can be used for both populating and retrieving information. The authors introduce two types

of embeddings, namely Word2Vec embeddings and Entity2Vec embeddings, which are trained on the table data to capture semantic similarities between words and entities. The embeddings are then used to generate a representation of each table, which can be used for tasks such as table completion and query answering.

Zhiyu et al. [15] proposes a deep contextualized language model based approach to return a ranked list of candidate tables in response to a keyword query. To achieve this, they leverage a pre-trained BERT model to embed both the query and table into a shared semantic space. As BERT has a limitation of 512 tokens on the size of input sequences, they use three content selectors to rank the terms of candidate tables and choose the best ones for the input. The model is fine-tuned on a large dataset of tables and queries, where each query is paired with the table that it best matches. During training, the model minimizes a ranking loss function that encourages the relevant table to be ranked higher than irrelevant tables. At query time, the user's query is encoded using the same pre-trained BERT model used during training, and the resulting query embedding is compared to the embeddings of all the tables in the dataset. The similarity between the query embedding and each table embedding is computed using cosine similarity, and the tables are ranked by their similarity scores.

Bogatu et al. [7] proposes a method for facilitating dataset discovery in data lakes as they argue that data lakes can be difficult to navigate and explore. Their proposed method leverages machine learning techniques to help users identify relevant datasets based on their search queries. The authors propose a two-stage approach for dataset discovery. In the first stage, the method uses a combination of keyword-based search and data profiling to identify datasets that are potentially relevant to the user's query. The authors use a keyword-based search approach to identify datasets that contain specific terms or phrases that match the user's search query. They also use data profiling techniques to analyze the metadata associated with each dataset, such as its schema, format, and size, to further refine the search results. In the second stage, the method uses machine learning techniques to rank the search results

based on their relevance to the user's query. Specifically, the authors use a combination of supervised and unsupervised learning techniques to build a relevance model that takes into account various features of the datasets, such as their metadata, content, and usage patterns. The relevance model is trained on a labeled dataset of queries and their corresponding relevant datasets, and is then used to rank the search results for new queries.

2.1.2 Table-Based Search

Before going into the details of existing works, it is worth to notice that many of them are using some form of a *matcher* to find columns that have been drawn from the same domain. Koutras et al. [43] identifies different types of matchers and categorizes them into 6 classes.

1. *Attribute Overlap Matcher*: it match columns with similar names, e.g. by using Leveneshtein Distance, and chooses those with similarity over a specific threshold as matches.
2. *Value Overlap Matcher*: two columns are drawn from the same domain if they have overlap in their values, e.g. measured using *Jaccard Similarity*. Those columns with similarity over a specific threshold are considered as matches.
3. *Semantic Overlap Matcher*: two columns are considered relevant if they have overlap in the set of entities they represent, for example after mapping their values to their entity types using a knowledge graph or a knowledge base.
4. *Data Type Matcher*: this matcher marks (ir)relevant columns based on their column types. For example, an integer column is irrelevant to a string column.
5. *Distribution Matcher*: it matches two columns if they have similar distributions for their values, e.g. by using *Jensen-Shannon Divergence*.
6. *Embeddings Matcher*: two columns with similar or same embedding representations of their values are considered relevant.

In many of the existing work, the first step is to extract features from elements of tables. These features may be extracted from table content, such as cell values, table schema like column headers and types, table context which includes the content around the table or the information on the webpage containing the table, statistics of the table such as cardinality and degree, and so on.

Vector-Based Approaches

In this thread of works, each table is represented as a vector by aggregating over vector representations of different elements of the table, and the relatedness or similarity of two tables is detected using functions over their vectors, e.g. dot product of vectors.

Das Sarma et al. [22] formalizes *entity complement* and *schema complement* as two concepts for finding related tables that augment an input query table by adding rows and columns respectively. For *entity complement*, they follow two different approaches. In one approach, they represent each entity of a table as a vector and consider the relatedness of two entities as the dot product of their vectors. Building on this, they define the relatedness of two tables as the average relatedness of their entity sets. In another approach, they directly represent the entities of each table as a single vector and define the relatedness of two tables as the dot product of their vectors. For adding columns, under *schema complement*, they use three different matchers to find tables related to a query table based on the overlap of their entity sets.

Fernandez et al. [27] proposes *SemProp* for automatically linking datasets by leveraging word embeddings. To apply this method to datasets, the authors first generate embeddings for each dataset in a given repository using the names and schemas of the datasets. They then construct a DAG where each dataset is represented as a node and edges between nodes represent the semantic similarity between the corresponding datasets. To calculate the similarity between two datasets, the authors propose a similarity measure based on the cosine similarity between their respective embeddings. When a user enters a search query, the method identifies the most relevant datasets by computing

the semantic similarity between the query and each dataset in the repository. The authors use a modified version of the TF-IDF algorithm to weigh the query terms and compute their similarity to each dataset. The most similar datasets are then ranked and presented to the user as potential matches.

In another work, Zhang et al. [89] uses hand-crafted features such as column headers and *column-to-column* and *table-to-table* similarity measures from *InfoGather* [82] and transfer all these features into three different semantic vector space representations: i) word-based embeddings, ii) graph-based embeddings and iii) entity-based embeddings. Finally, they propose an element-oriented table matching framework, based on the similarity between elements of the query table and candidate tables using their vector representations. For measuring element-wise similarity, they exploit different strategies such as cosine similarity between al pairs of semantic vectors.

ML-Based Approaches

Another group of methods, feed table features into a machine learning-based model such as classification models to determine if the table is related to the search query. Yakout et al. [82] proposes *InfoGather*, a holistic matching approach, to augment an input table with more rows and columns or to support the discovery of important columns of the input table using a corpus of webtables. It focuses on string columns and besides the traditional features such as columns headers and values used as bags of words for table-to-table similarity, they extract four more novel features from each webtable: i) context similarity, ii) table-to-context similarity, iii) URL similarity and iv) tuples similarity. Later they feed all features into a machine learning classification model to help calculating the matching level of each webtable to the input table.

Zhang et al. [85], extends *InfoGather* [82] and introduces *InfoGather+* to support numeric and time-varying columns as well. Handling these columns is challenging because they may have different units, scales or timestamps and this information is usually missing from column headers. They build a semantic graph by employing probabilistic undirected graphical models (i.e. *Markov* networks) to discover above mentioned information by searching for

semantically matching webtables. They rank these webtables based on their relevance to the input table and use them to do the entity augmentation task, i.e. adding rows. They exploit different indexes over both the semantic graph and the corpus of webtables to improve on efficiency.

Ahmadov et al. [2] does the *table imputation* task which is to fill the missing values of a table using lookup of webtables, training machine learning models over webtables or a combination of both. They create different inverted indexes over traditional features like column headers and use *Lucene* search engine to make the lookup operation efficient. They also train machine learning models like *RandomForest*, *J48*, *SimpleCart* and *SMO* over 27 extracted features of webtables and show that *RandomForest* yields the best results.

Zhang et al. [91] implements an interactive framework which incorporates specialized relevance-ranking criteria to support four different types of tasks: augmenting data, linking data, extracting ML features and data cleaning. Their framework combines different measures of table relatedness and exploits pruning and approximation strategies to return the top-k related tables to a given query table. The measures used include: i) table overlap in terms of rows and columns value or domain overlaps, ii) new information in terms of new rows and columns and iii) provenance similarity which capture semantic relatedness. They finally propose some novel indexes to make the search for top-k related tables efficient.

Cong et al. [21] proposes a method for semantic table union search. The authors argue that existing approaches for semantic table union search have limitations in terms of scalability, robustness, and generalization. They propose a new approach, *PYthonic table union search with Contrastive RepresentatiON learning (Pylon)*, that leverages contrastive representation learning to generate high-quality embeddings for the tables and their attributes. The method consists of two stages: representation learning and table union search. In the representation learning stage, the method uses a contrastive learning approach to learn embeddings for the tables and their attributes. The authors use a siamese network architecture to encode the tables and their attributes into vector representations, and they use a contrastive loss function to encour-

age the embeddings of similar tables to be closer in the embedding space. In the table union search stage, the method performs the table union by identifying potential matches between the tables based on their embeddings. The authors use a similarity measure that takes into account the embeddings of the tables and their attributes, as well as their structural compatibility, to identify the best matches. The authors evaluate their method on several real-world datasets and show that it outperforms existing approaches for semantic table union search in terms of accuracy and efficiency. They also show that their method is robust to noisy data and can generalize to unseen data sources. Overall, the proposed method leverages contrastive representation learning to generate high-quality embeddings for the tables and their attributes, which can improve the accuracy and efficiency of the semantic table union search.

Optimization-Based Approaches

Defining the table search task as an optimization problem is another approach in the literature. Nguyen et al. [54] introduces *diversified table selection* problem as selecting a diverse ranked list of tables and *structured table summarization* problem as summarizing the content of the returned tables meaningfully and concisely. Diversified table selection is defined as an optimization problem over the goodness score of the results which is obtained by using the relevance and similarity scores of webtables. This problem is known to be NP-Complete, hence they propose a greedy approach for it. For the structured table summarization problem, they tend to minimize the information loss by selecting a good set of representative tuples. Since this problem is also NP-Complete, they use a heuristic approach which makes different clusters for each group of similar tuples and returns representative tuples for each cluster to ensure diversity. They use schema matching methods to find tables with similar column headers and data similarity methods to find tables with similar cell values.

Probability-Based Approaches

Formulating the table search task as a probabilistic framework is another thread of research. Nargesian et al. [52] formalizes three matchers set do-

mains, semantic domains, and natural language domains to find unionable columns. under a set domains matcher it is checked if two columns have overlap in their set of values. For semantic domains, column values are mapped to knowledge base entities. Lastly, a matching based on natural language domains takes into account columns with natural language text as their values and uses word embeddings trained based on Wikipedia documents and cosine similarity to calculate their unionability. Having unionable columns at hand, they introduce a probabilistic framework to find candidate tables with a one-to-one mapping of columns to the query table. To find the best alignment between a candidate table and the query table, they reduce the problem to a weighted bipartite graph matching problem and try to find matchings of specific size. They also use inverted indexes to make the search efficient, showing their work can efficiently find the top-k candidate tables.

Matcher-Based Approaches

Some works rely purely on existing matchers to find related columns and build on them using similarity measures such as Jaccard to find related tables. Lehmberg et al. [49] introduces *Mannheim Search Join Engine (MSJ Engine)* with the goal of adding more columns describing entities of a given table in three main steps: i) table indexing, ii) table search and iii) data consolidation. For the indexing phase, they first find the main column of the corpus tables and then build two *Lucene* indexes, one over main columns values and the other over all column headers. For the searching step, they use two different methods on the main column of the input table: i) exact search over main column values and ii) similarity search over main column values using *FastJoin* matcher which extends the Jaccard similarity to allow fuzzy matching of tokens. Finally, they retrieve the top-k joinable tables and consolidate the input table using schema matching and data fusion techniques.

In another work, Sarabchi et al. [66] proposes a framework for augmenting a query table with new columns, a.k.a. table extension. The first step of their framework is to apply the table search task over a dataset of candidate webtables to get those webtables with at least one column characterizing a

query column. In order to find such columns, they use value overlap measure and choose those columns with a score over a threshold as of the same concept. In order to make the search efficient, they utilized an inverted index on the values of columns in the corpus.

Kassenov et al. [39] introduces an approach to augment a query table with more rows, a.k.a. table expansion. As the schema for webtables is not usually known, they focus on the content of the tables. They first split tables, both query table and candidate webtables, into entity-attribute binary relations using projection operation. Each entity-attribute relation is a set of key-value pairs with key being the primary key of tables and value being the value of another non-primary key column of the table. Then, they utilize co-occurrence statistics to retrieve best candidate relations and combine them into full candidate rows for the query table.

Other Table-Based Search Approaches

Zhu et al. [93] proposes a system called *JOining Search using Intersection Estimation (JOSIE)* for efficiently finding joinable tables in data lakes. JOSIE uses a novel algorithm called *Overlap Set Similarity Search (OSSS)* to compare the column names and values in different tables and identify potential join candidates. So, they reduce the problem of finding top-k joinable tables to the task of overlap set similarity. *JOSIE* utilizes a cost model which minimizes the number of set reads and inverted index probes and adapts to the data distribution which makes the search efficient.

Lehmberg et al. [47] shows that if table stitching is employed to combine small candidate tables into wider candidate tables, the table union search can be done more easily and more accurately. They first show that current matching tools, such as *T2K* which matches webtables to a knowledge base and *COMA* which is a schema matching system, both fail to produce acceptable results on datasets with small webtables. This is a real issue since the majority of webtables are small in size. To address this problem, they propose to combine webtables from the same webpage using schema matching methods (a.k.a. table stitching) to make a larger table for any further data discovery

task. They use three different matching approaches for the evaluation: i) label-based matcher, ii) value-based matcher and iii) duplicate-based matcher to stitch small webtables and show that stitching tables can have a strong effect on the performance of matching tasks.

Khatiwada et al. [40] proposes a novel method for semantic table union search. The authors argue that existing approaches for semantic table union search rely on column-level matchers and do not take into account the semantics and relationships of columns. The authors propose a new approach that leverages the relationships between tables in terms of semantic relationships between pairs of columns to identify potential matches and perform the table union. They call their approach SANTOS, which stands for "SemANtic Table UniOn Search." The method identifies column semantics by annotating each column to a set of entities in a KB or if there is no KB available, by putting attributes with similar semantics in the same group. It also assigns each pair of columns to a set of annotations by mapping subsets of columns values to a KB. Building on this two notions of column semantics and relationship semantics they build a semantic graph with columns as nodes and edges between two columns and their relationship semantics. Finally, they define top-k table union search as finding a subtree of the generated graph. Generally, the method consists of two stages: candidate generation and candidate selection. In the candidate generation stage, the method identifies potential matches between tables by leveraging their semantic attributes and their relationships with other tables. The authors use semantic embeddings after mapping columns to KB entities to represent the tables and their attributes, and they use graph-based algorithms to identify tables that are semantically related to each other. In the candidate selection stage, the method selects the best matches among the generated candidates by taking into account various factors, such as the semantic similarity between the tables and their structural compatibility. The authors use a ranking algorithm that combines these factors to score the candidate matches and select the best ones. The authors evaluate their method on several real-world datasets and show that it outperforms existing approaches for semantic table union search. They also compare

their method to other graph-based approaches and show that it achieves better precision and recall. Overall, SANTOS leverages semantic embeddings and graph-based algorithms to identify potential matches between tables and perform the table union by considering the relationships between tables.

2.1.3 Connection to Our Work

Our work is a hybrid approach utilizing the concepts of matcher-based and probability-based approaches. It takes as input a query table a corpus of candidate webtables and returns the most unionable webtables for the query. It is highly similar to Nargesian's work [52] in identifying unionable columns and tables. However, we apply preferences to the table search task which differentiates our work From theirs.

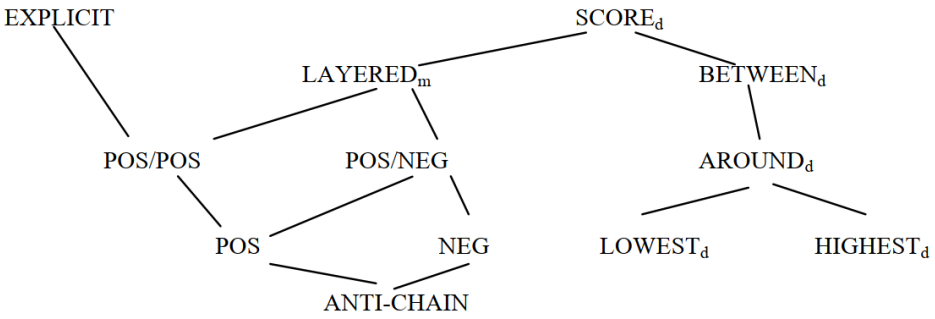
Only a few existing studies explore special cases of preferences. For instance, Khatiwada et al. [40] investigated the semantics and relationships of columns to identify unionable webtables more accurately. At first glance, their approach may seem similar to our work on dependent set preference. However, in dependent set preference, we aim to consider different combinations of column values for two unionable candidate webtables while they try to find more semantically relevant webtables. Thus, their approach can be regarded as a new preference in the context of our work.

2.2 Preferences

The other focus of this research is on preferences which is a different area than table search. Preferences are useful in the sense that they give some control over the search process to the user. This study discusses three popular preferences in the literature, skyline, diversity, and novelty, as well as a novel preference, dependent set. Although our work may apply to other preferences, we don't cover other preferences in the thesis to keep it focused.

Preferences have been widely studied in the literature as a solution to avoid returning empty or flooded results in response to a user query [41], [71]. Each preference is a soft-constraint, provided by the user, describing the way the

Figure 2.1: Base preference constructors proposed by [41]



results are computed or the properties that the results should follow. Kießling [41] proposes a hierarchy of base preference constructors for numerical and categorical data (Figure 2.1), based on which one can build more complex preferences. For example, $SCORE_d$ uses a scoring function and prefers the data with the higher (or lower) scoring function over the others. Building on this constructor, $BETWEEN_d$ prefers a numeric data in a specific range over another one out of the range. In this section we are going to talk about more complex preference queries in the literature. These preferences favor data that are more useful for the needs of users, but their implementations may be challenging, requiring efficient strategies.

2.2.1 Skyline

Skyline is a well-known preference extensively studied in the literature [17], [36]. It is commonly defined over data points of the same dimensionality and the aim is to find those data points which are not dominated by any other data point. A data point D_i dominates a data point D_j if it has as good as or better values on all dimensions and a better value on at least one dimension [16]. Kalyvas et al. [36] categorizes the works on Skyline into two groups: i) *index-based* methods and ii) *non index-based* methods.

Index-based methods

These methods [16], [42], [46], [56], [73] rely on pre-computed indexes on data to achieve a desired performance but with limited applicability to various

situations and suffering from the curse of dimensionality when the data points and consequently indexes are multi-dimensional.

Tan et al. [73] proposes *Bitmap* and *Index* algorithms to progressively find the skyline data points. In the *Bitmap* algorithm, which is a non-blocking approach, they encode and represent all the information of a data point required to check if it is skyline or not, as an m-bit vector. Then they exploit a bitmap structure to do the search over all data points using efficient bitwise *and* and *or* operations. The *Index* approach is a B^+-tree-based algorithm which transforms d-dimensional data points into a single dimensional space and partitions the dataset into d ordered lists. It then exploits a B^+-tree index with the actual data points as the leaves to return skyline points in batches. It is worth mentioning that in order to get all the skyline points, the algorithms should do a full scan of the dataset. They show that both algorithms have a quick initial response time and outperform the existing methods in total response time.

Kossmann et al. [42] introduces *Nearest Neighbor (NN)*, an online progressive algorithm which is useful specifically for interactive environments where the user needs to get some results early in the process or wants to modify the process by changing the distance function, for example. It uses R*-tree for indexing data points in order to prune many dominated points as early as possible. At the first step, *NN* will identify the data point with the minimum (or maximum) distance using a monotone distance function such as *l1-norm*. If we consider data points as 2-dimensional vectors, a skyline point divides the whole space into four partitions. All the data points in the top right (or bottom left) partition are being dominated by this point and can be removed safely. The bottom left (or top right) partition is empty based on the definition and so, the other two partitions will be added to the to-do list and the *NN* algorithm will be run over them recursively. However, the high number of IO access and the large size of to-do list are challenges of using this algorithm.

To address the issues with *NN*, Papadias et al. [56] proposes *Branch and Bound Skyline* which is a progressive approach supporting user preferences during its process. The approach offers an IO optimal solution by accessing

only those partitions with potential skyline points. It also needs a limited main memory to operate and can be used on any subset of dimensions without the necessity to pre-compute anything other than the R-tree index. The authors also introduce some new variants of skyline query such as *constrained skyline*, *k-dominating skyline* and *approximate skyline*. The authors show that their approach is applicable to diverse scenarios with less limitations while outperforming them in terms of the IO cost.

Lee et al. [46] presents the *Z-SKY* framework which computes skyline based on *Z-order* curves. They exploit a *ZBtree* index structure by combining *Z-order* curve index structure with a B^+-tree data structure. The *Z-order* curves, as a monotonic dimension reduction technique, clusters data points based on their position on Z-ordering, transforming data points into a one-dimensional space. Their results show effectiveness of *Z-SKY* compared to BBS [56] method.

A big issue with existing methods is that as the number of dimensions increases or the dataset gets bigger, the indices do not perform well and the overall performance drastically decreases. Choi et al. [16] introduces *Hash Index-based Skyline (HI-Sky)* approach to solve this issue by exploiting the fusion of a hash index and a grid partitioning ,*HashGrid*, over dataset to efficiently perform the skyline computations. They assign grid location address (*GLAD*) to each grid partition and perform partition-wise comparisons to prune dominated partitions without even accessing their whole data points to compute the skyline. They show that their approach beats *Z-SKY* and *BBS* [56] by generating the index structure faster and executing skyline queries more efficiently over high dimensionality datasets.

Non Index-based methods

These methods [4], [8], [18], [30] are more generic since they are not limited to the existence of any specific index structure. Although their performance is not as good as index-based methods, they are not suffering from the curse of dimensionality and can be applied in many situations.

Borzsony et al. [8] proposes two algorithms *Block Nested Loops (BNL)*

and *Divide and Conquer (D & C)* to find skyline points in a large dataset. *BNL* allocates a buffer of limited size in main memory for keeping skyline data points in a self-organizing list. It iterates over all the data points and compares them against current skyline data points in the buffer. A data point D_i will be pruned if it is dominated by at least one skyline data point in the buffer; otherwise, any skyline data point dominated by D_i will be removed from the buffer and D_i will be added to the list as a newly discovered skyline point. A temporary overflow disk is used when the buffer is full and a new non-dominated point is found. After reading all data points, it is guaranteed that those in the buffer are in fact skyline data points, but such a claim cannot be made for those data points in the temporary disk. They consider the temporary disk as a new dataset and do the process over it again and so on and so forth until there is no data point in the temporary disk. The performance of *BNL* is sensitive to the number of skyline data points, the dimensionality of data points and the underlying distribution of data points. They show that it works well for data points with a uniform distribution and less than 5 dimensions. The D & C algorithm takes another approach and based on α-quintiles of the dataset along a specific dimension, recursively divides the dataset into m partitions each expected to be fit in the memory. The skyline data points of each partition are extracted using traditional comparison of data points and all the partitions' skyline data points are merged at the end by eliminating the dominated ones. These algorithms are not progressive meaning that they do not return any skyline data point unless the whole search process is done completely. D & C does not suffer from the curse of dimensionality and it is not as sensitive to data points distribution as the BNL.

To improve BNL and reduce CPU-boundedness, Chomicki et al. [18] introduces *Sort-Filter-Skyline (SFS)* algorithm which pre-sorts the data points based on a monotone scoring function in an ascending order. The monotone function might be the sum of each data point over all the dimensions. Based on this order, if a data point dominates another data point, it will be ranked higher in the sorted list and will be visited first. SFS uses a buffer similar to BNL's buffer to keep skyline data points which will be gathered by iterating

over the sorted list of data points and identifying those that are not dominated by any other skyline data point in the buffer. Following this approach, they might find skyline points progressively and efficiently by reducing the number of dominance checks.

Godfrey et al. [30] applies some optimizations on the SFS [18] algorithm to improve its performance. Their algorithm, *Linear Elimination Sort for Skyline (LESS)*, is a generic maximal-vector external algorithm which sorts the data points based on a monotone scoring function and uses a *Skyline Filter(SF)* window just like SFS to find the skyline data points. LESS presents improvements to the SFS algorithm, such as the use of a heap data structure to speed up the sorting process and the application of randomized sampling techniques to reduce the computational complexity. They show that the average performance of *LESS* is linear in the number of data points and it improves the existing algorithms significantly.

Bartolini et al. [4] proposes *Sort and Limit Skyline algorithm (SaLSa)* as an improvement over *SFS* [18] and *LESS* [30] by avoiding scanning all the data points in the dataset in order to get skyline points. Like the base *SFS* and *LESS* algorithms, *SaLSa* does not use any pre-computed index structure over data points and it sorts the dataset based on a monotone scoring function. For sorting, they exploit the sorting machinery of a relational engine to order data points in a way that only a subset of them need to be examined to get to skyline data points. In other words, the novelty of their work is that they do not perform dominance checks over all the data points of the dataset although they need to iterate over all data points to sort them. Their experiments show that their approach is effective in the action.

2.2.2 Diversity

A search query can be a set of keywords or a table, and both may be ambiguous with more than one interpretation [62], [92]. Also, some interpretations may be more important than others. For example, 'Apple' may refer to a fruit or the Apple company. Returning results all about one interpretation may lead to user dissatisfaction as another interpretation of the query may be of interest.

Even if the search query has no ambiguity, e.g. 'BMW', returning results all about same entities may not be interesting for a user who searches for new entities. For example, instead of returning results all about one specific model of 'BMW' vehicles, it may be more interesting to return results from different models [92]. The idea behind diversity preference is to return results from different interpretations or about different entities of the search query in order to satisfy a wider range of users.

Zheng et al. [92] proposes the general steps it takes for an approach on diversification of results to return the diverse list of k results as follows. Given a search query, the first step is to find all the relevant results and rank them based on their relevancy to the query. The next step is to diversify the ranked list of relevant results and return the top-k ones to the user. The early works [81] on returning top-k diverse candidates from a large dataset, retrieve $c \times k$ top relevant candidates, where c is a constant greater than 1, and search over this smaller dataset for the top-k diverse candidates. Others, execute multiple queries with different constraints to get diverse candidates. Another thread of works, exploit sampling techniques to retrieve a smaller representative version of the large dataset and return the top-k diverse results of the smaller dataset [62]. Clustering techniques are another option that some works apply to get a diverse list of candidates by choosing them from different clusters [75].

Zhang et al. [92] has a survey on query diversification methods which groups them into three categories: i) *Content-Based Diversification*, a.k.a. similarity-based diversification, ii) *Intent-Based Diversification*, a.k.a. coverage-based diversification, and iii) *Novelty-Based Diversification*. The third category is considered as a separate preference in the literature, known as *novelty*, and so we are talking about it in the next section (Section 2.2.3).

Content-Based Diversification methods

This group of methods [1], [34], [75], [76], [78], [84] assume that the search query has no ambiguity, and they choose candidates that are as dissimilar as possible to each other while they all are relevant to the search query. The general idea is to first define a diversity scoring function over the list of top-k

results. Usually, maximizing or minimizing this function is computationally hard [1], [78] which leads to using greedy algorithms for finding an optimal solution.

Yu et al. [84] selects the top-k relevant candidates as the initial list and exploits two ideas to go forward. The first idea, *Swap* method, is to iterate over the rest of candidates and try to swap any unselected one with a candidate in the top-k list if it makes an improvement in the objective function. This process continues until no other candidate remains unchecked. The other idea, *BSwap* method, is to choose the candidate that makes the biggest improvement on the objective function in each step until there is none.

Van Leuken et al. [75] proposes a clustering technique which first divides all candidates into k clusters based on *k-medoid* algorithm and then choose one candidate from each cluster to build the top-k diverse results.

Vee et al. [76] exploits inverted lists and a B^+-tree to skip similar answers and propose two algorithms, *one-pass* and *probing*. In the one-pass algorithm, which the *WAND* approach is used to iterate over the lists, only one iteration over data is allowed to get the top-k diverse results. In the probing algorithm, they have the option to go over data for a few times. Their experiments show that both algorithms are scalable and efficient.

Agrawal et al. [1] proposes a probabilistic submodular objective function for finding top-k diverse results, taking into account not only their diversity but also their relevance. They show that finding the exact results to maximize the objective function is an NP-Hard problem. So, they use a greedy approach, *IA-SELECT*, to find an optimal set of results. Their greedy algorithm retrieves the top-k results of some classical ranking algorithm and reorder them in a way that maximizes the objective function. Their evaluations show the effectiveness of their greedy approach on different metrics.

In a similar work, Vieira et al. [78] presents *DivDB* which is a probabilistic framework that considers relevancy and diversity of data items in order to get a *k-similar diversification set*. In particular, it accepts any distance and similarity function in the formulation, which makes it applicable in many situations by adjusting those two functions. Hurley et al. [34] models the two

objective functions *diversity* and *relevancy* as a binary optimization problem.

Intent-Based Diversification methods

This group of works [12], [20], [62], [65] focus on the ambiguity of the search query and look for candidates from different facets of the search query. The general idea is on covering all the different interpretations of the search query. Some works take one step further and try to choose candidates in a weighted fashion. The idea is that each facet of the search query has a level of importance different from others. This importance can be represented as a weight which results in choosing more candidates from facets with higher weights.

Radlinski et al. [62] introduces a framework based on *query-query* reformulation, which is the process of modifying the original query or generating new queries based on the user's input or feedback, and presents three methods over this framework to increase the diversity of the top-k results of a web search query. They first generate a set $M(q)$ of k queries that are related to search query but at the same time different from it using query logs of a common web search engine. These queries are intended to cover all the facets of the search query. The first method, *Most Frequent*, defines $M(q)$ as the set of queries that are observed the most after the search query. The second one, *Maximum Result Variety*, follows a greedy approach and uses the same steps as the first method but at each step tries to select a frequent query that is different from the selected ones so far. Finally, the *Most Satisfied* builds $M(q)$ by choosing queries that are rarely followed by the search query using some thresholds. Their evaluations show a promising improvement on the diversity of the results.

Clark et al. [20] addresses the issue of ambiguity that may happen in information retrieval question answering. They focus on an evaluation framework which treats questions and answers as 'information nuggets'. Based on their framework, it is better to have distinct results representing different concepts to cover all interpretations and to return more results about the concepts which are more interesting than others.

Rafiei et al. [63] views search diversification as a means of risk minimization

and models the problem as an expectation maximization task. The authors develop a weight vector, W, to assign weights to search results, with the sum of all weights adding up to one. They also introduce the concept of portfolio variance, defined as $W^T CW$, where C represents the covariance matrix of the result set. Their goal is to minimize portfolio variance while maintaining a fixed level of expected relevance. The authors find that their proposed algorithm yield more diverse search results than Google. In a test with 156 random queries, their algorithm retrieves between 14% to 38% more relevant results in the top five, while still maintaining a precision that is comparable to Google's.

Santos et al. [65] introduces *eXplicit Query Aspect Diversification (xQuAD)* which is a probabilistic framework for result diversification of web search queries. It ranks documents in a way that the rank of documents with coverage of uncovered facets of the search query are boosted. The goal is to cover as many facets as possible with a trade-off between relevance and diversity. They exploit query reformulations, which is the process of modifying a user's initial search query in order to retrieve a more relevant set of search results, provided by existing search engines to uncover different facets of the search query.

Capannini et al. [12] proposes *OptSelect*, which follows three steps to re-rank the original results of a search query. In the first step, it checks if there are different facets of the search query. For the second step, it retrieves documents related to different facets by mining specializations from query logs. Lastly, in the third step, it uses a probabilistic framework to maximize an objective function and get the most diverse list of results. They show that their work is effective, efficient and outperforms *IS-SELECT* [1] and *xQuAD* [65].

2.2.3 Novelty

This line of work is related to the work of diversification with a focus on getting results with novel information for the search query, compared to those previously retrieved. Having duplicate information in the results is not delightful for the user. Novelty is the preference which takes care of this situation and

returns diverse information to the user. The novelty of a document refers to how different it is from the previously seen documents [13]. Based on a survey done by Ghosal et al. [28], we categorize novelty studies into three classes: i) *sentence-level novelty*, ii) *document-level novelty* and iii) *diversity-based novelty*.

Sentence-Level Novelty

Early works focus on sentence-level novelty detection [45], [68], [69]. For a given topic and an input list of relevant documents, the goal of these techniques is to find sentences that are both relevant to the topic and carry new information about it. They usually exploit language and vector space models alongside cosine similarity to find the novel relevant sentences.

Document-Level Novelty

This thread of work takes one step further and concentrates on document-level novelty detection. Yang et al. [83] first uses a supervised learning algorithm to put online document streams in pre-defined topic categories and then exploits *Named Entities* to detect the novelty of documents in each topic. Tsai et al. [74] proposes a *document-to-sentence (D2S)* framework which segments a document into sentences, determines the novelty of each sentence and finally computes the novelty of each document by aggregating the novelty of it's sentences.

Karkali et al. [38] exploits IDF scoring function to score documents based on their novelty. They rely on documents statistics to avoid computing the similarity or distance of a document with the rest in order to compute its novelty. Dasgupta et al. [23] apply information theoretic measures and term-domain relevance to automatically compute *innovativeness* score of a document with respect to a collection of documents. A document is considered novel if it contains a unique topic not presented in other documents or a rare topic mentioned only in a few documents. They also use IDF score to compute the specificity of each of the individual terms in a document.

Diversity-based Novelty

Novelty in information retrieval is usually defined as a trade-off between relevancy and diversity of the content of documents. Clark et al. [20] addresses the issue of redundancy that may happen in question answering. Their evaluation framework treats questions and answers as sets of independent 'information nuggets' and the relevance as a function of common nuggets between questions and answers. They consider a document as relevant if it contains previously unreported nuggets useful to the user. They develop a measure based on *Normalized Discounted Cumulative Gain (nDCG)* to formalize their work.

Qin et al. [61] proposes a framework that can extend most existing approaches on returning top-k relevant results for a given query to return top-k diverse results. Furthermore, the authors propose three different greedy algorithms for returning top-k diverse results. The first algorithm, called div-astar, is based on A*. The second algorithm, div-dp, breaks down the results into components, which are then independently searched using div-astar and combined using dynamic programming. Finally, the third algorithm, div-cut, further divides the generated results using cut points and combines them using advanced operations.

2.2.4 Connection to Our Work

Our work introduces preferences to the table search task. We mainly focus on skyline, diversity and novelty, three preferences from the literature and a novel preference, referred to as dependent sets, which is useful in the context of our work. As we will show in Section 2.2, no work is done on applying skyline on table search. For the diversity, our work is very similar to the underlying frameworks of Nguyen [54] and Vieira [78]. We have an objective function which computes the goodness scores of the chosen top-k candidates while finding a trade-off between relevance and diversity. Regarding novelty, our work takes a similar approach to diversity, as they both diversify the results, by offering an objective function as a trade-off between relevance and novelty. For both of these preferences, we exploit a greedy approach like

Swap method [84], which tries to improve the goodness score by swapping the unselected candidate with the best improvement.

Chapter 3

Methodology

In this chapter, we first introduce column and table unionability scores, inspired by Nargesian et al. [52], and then introduce our TUS operation based on those scores. For the TUS operation, we propose two efficient implementations based on the WAND algorithm. Next, we present our preferences, which include four major ones: skyline, diversity, novelty, and dependent set. Later, each preference is discussed in terms of how it can be efficiently evaluated.

3.1 Table Union Search (TUS)

To find unionable candidate webtables for a query table, one may first find unionable candidate columns for each query column. The unionability of two columns indicates if they are drawn from the same domain, and this notion can be defined in many different ways. A simple approach is based on the similarity or the value overlap between the two columns. The more the similarity between their values, the higher the chance of them being unionable. If S_1 and S_2 denote the set of values in two columns, the similarity between the two sets may be defined in terms of the *Jaccard* similarity, expressed as $|S_1 \cap S_2|/|S_1 \cup S_2|$. We refer to the score that such approaches calculate to represent the degree of unionability of two columns as their *Column Unionability Score (CUScore)*. For the rest of this thesis, unless explicitly stated otherwise, we use *Jaccard* to calculate CUScore of two columns.

Definition 3.1 (Unionable Pair)**.** Two columns c_i and q_j are called a unionable pair, shown as $c_i \rightarrow q_j$, if their CUScore is over some confidence level (like 0.9).

Consider query table Q, with columns q_1, q_2, ..., q_n and a candidate webtable C, with columns c_1, c_2, ..., c_n, both of degree n. We refer to the set of all possible unionabile pairs between the columns of two tables as $UP(C, Q)$ and use it to define an alignment between the two tables (Definition 3.2).

Definition 3.2 (Alignment)**.** An *Alignment* between a query table Q with columns $q_1, q_2, \ldots, q_n$ and a candidate webtable C with columns $c_1, c_2, \ldots, c_n$, both of the same degree n, is a subset of unionabile column pairs in $UP(C, Q)$ which form a bijective mapping between the columns of C and Q and is shown as $a(C, Q)$. The size of $a(C, Q)$, shown as $|a(C, Q)|$, is the number of unionable pairs it contains.

In other words, an alignment is a one-to-one mapping between columns of both tables with each mapping representing a unionabile column pair. Note that based on this definition, C and Q may have multiple alignments of size n, each made up of different unionabile pairs. For example, one alignment may be composed of unionabile pairs $c_i \rightarrow q_j$ where $i = j$, $1 \leq i, j \leq n$ and another one of unionabile pairs $c_i \rightarrow q_j$ where $i + j = n$, $1 \leq i, j \leq n$.

But in real world, two tables C and Q may have different degrees. Consider query table $Q(q_1, q_2, \ldots, q_n)$ of degree n and candidate table $C(c_1, c_2, \ldots, c_m)$ of degree m where $n \neq m$. If we denote with u the number of unionable column pairs between C and Q, then $0 \leq u \leq n \times m$. The minimum, $u = 0$, happens when two tables are completely irrelevant and there is no unionable column pair $c_i \rightarrow q_j$ between the two tables, i.e. $UP(C, Q) = \emptyset$. The maximum, $u = n \times m$, is when every column c_i of C is unionable with every column q_j of Q, $UP(C, Q) = \{c_i \rightarrow q_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. Now imagine any subset of

these unionable pairs with only one constraint that no column c_i or q_j should appear in more than one unionable pair, i.e. conflicts are not allowed in any direction. We can think of this subset as an alignment between a projection operation over Q and a projection operation over C. We refer to the set of all these possible alignments that result from projection operations over C and Q as the *Alignment Set* of the two tables.

Definition 3.3 (Alignment Set)**.** An *Alignment Set* between a candidate webtable $C(c_1,\ c_2,\ \ldots,\ c_m)$ and a query table $Q(q_1,\ q_2,\ \ldots,\ q_n)$, denoted as $A(C,\ Q)$, is the set of alignments between all possible projections of the same degree over C and Q. ${A^c}_S(C,\ Q)$ denotes only the subset of alignments in $A(C,\ Q)$ which are between projections of degree c and with only mappings over a subset S of query columns.

We can interpret each unionable pair as an independent event and their CUScore as the probability of that event occurring. Following this interpretation, an alignment can be interpreted as the occurrence of multiple independent events together. Consequently, the unionability score of an alignment $a(C,\ Q)$ is the product of the probabilities of events in $a(C,\ Q)$ assuming independence.

Definition 3.4 (Alignment Unionability Score (AUScore))**.** The unionability score of an alignment $a(C,\ Q)$ between a candidate webtable C and a query table Q is the product of column unionability scores of all unionable pairs in $a(C,\ Q)$, i.e.

$$AUScore(a(C,\ Q)) = \prod_{(c_i,\ q_j) \in a(C,\ Q)} CUScore(c_i,\ q_j). \tag{3.1}$$

Note that based on Definition 3.4, if we add a unionable pair to an alignment in order to build a bigger one, the probability decreases. This makes the comparison of alignments of different sizes challenging. To make all alignments comparable, for all alignments of the same size c, we make a continuous

probability distribution over their AUScores using a big sample dataset. Since we have alignments with sizes from 1 to n, where n is query table's degree, we end up with n different probability distributions. Then, if for each alignment $a(C, Q)$ we compute $CDF(AUScore(a(C, Q)))$ over the corresponding distribution for $|a(C, Q)|$, we can bring all the AUScores to the same scale and make alignments of different sizes comparable [52]. We define this updated score after applying CDF function as the *Alignment Goodness Score (AGScore)* of the alignment (Definition 3.5).

Definition 3.5 (Alignment Goodness Score (AGScore))**.** The goodness score of an alignment $a(C, Q)$ between a candidate webtable C and a query table Q is the cumulative distribution function (CDF) of its AUScore using the probability distribution function computed over alignments of size $|a(C, Q)|$, i.e.

$$AGScore(a(C, Q)) = CDF_{|a(C, Q)|}(AUScore(a(C, Q))). \tag{3.2}$$

We know that for alignments of the same size, the higher the AUScore of the alignment, the higher the unionability of the pairs of columns it maps and the more interesting it is for the task under consideration. Generally, considering alignments of different sizes too, the higher the AGScore, the more unionable and hence more interesting the alignment is. The alignment with the maximum AGScore, referred to as the *max-Alignment*, represents the best mapping between the two tables. We define *Table Unionability Score (TUScore)* of two tables C and Q as the maximum goodness score of all alignments in the alignment set $A(C, Q)$ (Definition 3.6).

Definition 3.6 (Table Unionability Score (TUScore))**.** The unionability score of a candidate webtable C with respect to a query table Q is the maximum goodness score of all alignments in the alignment set $A(C, Q)$.

$$TUScore(C, Q) = \underset{a(C, Q) \in A(C, Q)}{\operatorname{argmax}} AGScore(a(C, Q)). \tag{3.3}$$

Consequently, we use $TUScore^c{}_S(C, Q)$ to denote the table unionability score over alignments in $A^c{}_S(C, Q)$.

Now that we have all the tools, we define table union search as the task that returns the top-k most unionable candidate webtables for a query table Q (Definition 3.7).

Definition 3.7 (Top-k TUS)**.** Given a query table $Q(q_1, q_2, \ldots, q_n)$ with degree n, Top-k TUS is the task of finding the top-k candidate webtables in the corpus with the highest unionability score to Q.

We next discuss how one can can perform top-k TUS efficiently. A naive approach involves examining every candidate webtable in the dataset, finding its best alignment with the query table under consideration, and ranking the candidates according to their table unionability scores. However, this approach is neither efficient nor scalable for large datasets. We propose an efficient algorithm for TUS in the next section. Further, some heuristics are applied in order to make the returned webtables more interesting for the user over different tasks.

3.2 Base TUS Algorithm

An obstacle for processing TUS operation efficiently is the examination of a huge space of candidate webtables. If we manage to prune many of these candidate webtables without fully examining them, that will improve the efficiency of the task. Existing works [52] try to overcome this challenge and offer an efficient unsafe ranking approach by sacrificing the accuracy and returning an approximately good results. This trade-off, however, may not works as expected in the presence of preferences. Those high-ranking candidate webtables being missed for the approximate results due to the trade-off, can be a great sources of information for some preferences.

We need to have a safe ranking approach which guarantees to return the absolute high-ranking candidate webtables without sacrificing efficiency. We need the safe ranking as those candidate webtables that are missed in an unsafe ranking algorithm, may indeed be the best candidates for some of the preferences. In other words, some preferences may prefer those candidate webtables with a small unionability as they may offer more new values to the query table.

We utilize the WAND algorithm [9] to prune candidate webtables early in the process without fully examining them. The WAND algorithm is a powerful document-at-a-time technique used for efficiently processing of conjunctive queries over inverted indexes. The algorithm was first introduced in a paper by Broder et al. [9] and has since become a popular choice for search engines that deal with large volumes of data. The WAND algorithm achieves its efficiency by using a threshold score to limit the number of documents that need to be examined, drastically reducing the search time. The algorithm also maintains an upper-bound score for each document that serves as a filter to skip documents that cannot possibly be in the top-k results, further reducing the number of documents that need to be scored.

WAND uses a "Weak And" approach to score documents. It performs a weighted sum of scores for each query term that appears in a document, where the weight is based on the document frequency of each term. The algorithm then performs an "and" operation on the resulting scores for each document to determine the final score. The top-k documents with the highest scores are returned as the search results. It has been shown to be highly effective at processing conjunctive queries on large-scale search engines. In comparison to other state-of-the-art algorithms, the WAND algorithm outperforms in terms of efficiency and accuracy. The algorithm's ability to handle large volumes of data and produce exact search results has made it a popular choice for search engines, enabling them to provide fast and reliable search results to their users. The WAND algorithm's ability to balance speed and accuracy has made it an essential tool for information retrieval, providing an excellent solution to efficiently processing conjunctive queries over inverted indexes.

There are some challenges in applying WAND to table union research. First, it needs to be adapted to the domain of table union search, where the query and candidates are not documents, but webtables. If we consider candidate webtables as documents and each of their columns as a term, then we can build inverted indexes for the query table. Each query column will be considered as a query term equipped with a posting list. The posting list of query column q_i is the ordered list of candidate webtables having at least one unionable candidate column with q_i with their column unionability score with q_i as their weight. The candidate webtables for each posting list are ordered based on their id with C_1 being first, C_2 being second and so on and so forth. Following this interpretation, the "and" operation over terms must be translated into a multiplication of term weights in order to reflect alignment and table unionability scores. Consequently, the upper-bound threshold used by WAND will be the maximum possible unionability score we can expect from a candidate webtable. Second, any function that is going to be used instead of "and" operation to calculate the score of a candidate webtable should be a monotone function in order for the pruning to works perfectly. Our alignment and table unionability scores are multiplication functions which indeed are monotone, but this may not be the case for some preferences.

This implementation, referred to as *TUS* in the rest of this thesis, is the base algorithm we use in our evaluations (Algorithm 1). We may apply some constraints to TUS to make the results more desirable for the follow-up operations the user intends to do. For example, returning only candidate webtables with an alignment over the primary key of the query table or those with an alignment over all the query columns may help more with follow-up operations like adding new rows or finding the missing values. For the rest of this thesis, we refer to TUS with additional constraints as *TUS+*.

Our hypothesis is that applying preferences to TUS can make significant improvements over some down-stream tasks. In the following section, we provide a running example and demonstrate the step-by-step process of executing the TUS operation to illustrate how each component of this operation functions.

Algorithm 1 WAND_Algorithm: returns top-k candidate webtables that match the query table Q with query columns qcs using the $inverted\_index$.

Input: qcs, inverted_index, k
Output: top-k webtables that match the query table
initialize query column pointers: pointers ← qc: inverted_index[qc].begin() for $qc \in qcs$
while not all pointers have reached the end of their lists **do**
    min_cw_id = the minimum candidate webtable among the current positions of pointers
    for all $qc \in query\_columns$ **do**
        if $min\_cw\_id < pointers[qc].current\_cw\_id$ **then**
            Advance the qc's pointer until it reaches a point where $pointers[qc].current\_cw\_id >= min\_cw\_id$
        end if
    end for
    ccs = get all the candidate columns for webtable min_cw_id
    $alignments$ = use bipartite_matching or greedy algorithms to get all alignments of min_cw_id using ccs
    for all $a \in alignments$ **do**
        if $AGScore(a) >$ k-th item in the top-k list **then**
            add a, min_cw_id to the list
            remove the webtable with the smallest score from the list
        end if
    end for
end while
Return the top-k list, sorted in descending order of score

3.3 Running Example

Consider the query table Q presented in Table 3.1 and candidate webtables $C_1 - C_9$ presented in Table 3.2. The posting lists of each query column of Q, computed based on the Jaccard similarity, is shown in Table 3.3. The posting list of a query column q_i is an ordered list of candidate webtables with at least one unionability pair with q_i based on their id with their CUScore with q_i as their weight.

Query Table Q			
q1 [movie]	**q2 [actor]**	**q3 [year]**	**q4 [distributor]**
Avengers: End Game	Robert Downey Jr.	2019	Walt Disney Studios Motion Pictures
Avengers: End Game	Chris Evans	2019	Walt Disney Studios Motion Pictures
The Avengers	Robert Downey Jr.		Walt Disney Studios Motion Pictures, Walt Disney Pictures, Paramount Pictures
The Avengers	Chris Hemsworth	2012	Walt Disney Studios Motion Pictures, Walt Disney Pictures, Paramount Pictures
Pulp Fiction	Samuel L Jackson	1994	Miramax

Table 3.1: Query table Q

Candidate Webtable C1		
c11 [movie]	**c12 [actor]**	**c13 [year]**
Avengers: End Game	Robert Downey Jr.	2019
The Avengers	Robert Downey Jr.	2012
The Godfather	Al Pacino	1972

Candidate Webtable C2		
c21 [movie]	**c22 [actor]**	**c23 [role]**
Avengers: End Game	Chris Hemsworth	Thor
The Avengers	Chris Evans	Captain America
Pulp Fiction	John Travolta	Vincent Vega

Candidate Webtable C3		
c31 [movie]	**c32 [actor]**	**c33 [director]**
Pulp Fiction	Samuel L Jackson	Quentin Tarantino
The Avengers	Mark Ruffalo	Joss Whedon
Extraction	Chris Hemsworth	Sam Hargrave

Candidate Webtable C4	
c41 [movie]	**c42 [actor]**
Avengers: End Game	Robert Downey Jr.
The Avengers	Chris Hemsworth

Candidate Webtable C9	
c91 [company]	**c92 [founder]**
Walt Disney Studios Motion Pictures	Walt Disney
Sony Pictures	Warner Bros.
Paramount Pictures	William Hodkinson
Universal Pictures	Carl Laemmle
Yash Raj Films	Yash Chopra

Candidate Webtable C5		
c51 [book]	**c52 [author]**	**c53 [year]**
The Nickel Boys	Colson Whitehead	2019
Gone Girl	Gillian Flynn	2012
The Chamber	John Grisham	1972

Candidate Webtable C6	
c61 [actor]	**c62 [age]**
Al Pacino	82
Meryl Streep	73
Daniel Day-Lewis	65
Samuel L Jackson	73

Candidate Webtable C7		
c71 [country]	**c72 [city]**	**c73 [language]**
USA	New York	English
Canada	Edmonton	English
Iran	Tehran	Persian
Iran	Ilam	Kurdish

Candidate Webtable C8	
c81 [movie]	**c82 [budget]**
The Avengers	220m
Avatar	237m
Casablanca	1.03m
The Artist	15m

Table 3.2: Candidate webtables $C_1 - C_9$

Using these posting lists, we can find alignments of different sizes between each candidate webtable and Q. In Table 3.4, for each possible subset of

query columns, the top-2 unionable candidate webtables and their AUScores and AGScores are shown. Notice that for some subsets of query columns, e.g. $\{q_1, q_2, q_3\}$, there are less than 3 candidate webtables with an alignment. Since each candidate webtable of this example has at most one alignment for each possible subset of query columns, following the Definition 3.6, the TUScores of candidate webtables would be the same as their AGScores.

QC	**Posting List**
q_1	$c_{11} : 0.5$, $c_{21} : 1.0$, $c_{31} : 0.5$, $c_{41} : 0.667$, $c_{81} : 0.167$, $others : 0.0$
q_2	$c_{12} : 0.2$, $c_{22} : 0.4$, $c_{32} : 0.4$, $c_{42} : 0.5$, $c_{61} : 0.143$, $others : 0.0$
q_3	$c_{13} : 0.4$, $c_{53} : 0.4$, $others : 0.0$
q_4	$c_{91} : 0.125$, $others : 0.0$

Table 3.3: Posting lists of query columns of Q. QC is the query column;

Based on the Definition 3.7 and the results in Table 3.4, the top-2 candidate webtables are shown in Table are: 1) C_2 with score 0.905 and max-alignment $\{c_{21} \rightarrow q_1\}$ and 2) C_4 with score 0.854 and max-alignment $\{c_{41} \rightarrow q_1\}$.

Subset	**Top-2 AUScores**	**Top-2 AGScores**
$\{q_1\}$	$C_2 : 1.0$, $C_4 : 0.667$	$C_2 : 0.905$, $C_4 : 0.854$
$\{q_2\}$	$C_4 : 0.5$, $C_2 : 0.4$	$C_4 : 0.808$, $C_2 : 0.767$
$\{q_3\}$	$C_1 : 0.4$, $C_5 : 0.4$	$C_1 : 0.767$, $C_5 : 0.767$
$\{q_4\}$	$C_9 : 0.125$	$C_9 : 0.491$
$\{q_1, q_2\}$	$C_2 : 0.4$, $C_4 : 0.334$	$C_2 : 0.786$, $C_4 : 0.759$
$\{q_1, q_3\}$	$C_1 : 0.2$	$C_1 : 0.674$
$\{q_2, q_3\}$	$C_1 : 0.08$	$C_1 : 0.499$
$\{q_1, q_2, q_3\}$	$C_1 : 0.04$	$C_1 : 0.555$
others	None	None

Table 3.4: Top-2 AUScores and AGScores for all possible subsets of query columns over candidate webtables in Table 3.2.

Rank	**CW**	**Alignment**	**TUScore**
1	C_2	$\{c_{21} \rightarrow q_1\}$	0.905
2	C_4	$\{c_{41} \rightarrow q_1\}$	0.854

Table 3.5: Top-2 TUS of Q; CW stands for candidate webtable;

In the next section, we will present four major preferences and add them to the TUS operation. Each one of these preferences introduce some challenges. Our goal is to address these challenges and propose an efficient algorithm for each preference.

3.4 Preferences

TUS looks at all the candidate webtables using the TUScore as the scoring function with the goal of finding the most unionable ones to a query table. A few issues with this process may lead to users dissatisfaction. For example, the list of returned candidate webtables is fixed and independent of the follow up operations that users may have in mind. Users may prefer some unionable webtables over the others simply because they offer alignments over some important subset of query columns, for instance. In this section, we present some preferences to deal with the problems of TUS and return more interesting webtables suitable for different users' needs. In particular, we provide four major preferences skyline, diversity, novelty and dependent set and some minor ones for the user to take advantage of in customizing and enriching TUS operation. For example, a preference may assign different weights to different query columns or some column may be preferred over the others. A preference may also consider the dependency of query columns and take into account their values together. Although for some of the proposed preferences, there is a workaround solution using only TUS, these approaches are usually inefficient, requiring more efforts and resources.

3.4.1 Skyline

Consider a query table Q (Table 3.1) with four columns 'movie', 'actor', 'year' and 'distributor'. Under some setting, all query columns may be important, and returning the most unionable candidate webtable for each query column may be an interesting source of information necessary for follow up operations. Taking one step further, the user may also be interested in having the best candidate webtables unionable over each subset of query columns, e.g. $\{movie, actor\}$. This may give the user a great tool for doing extra follow-up operations, such as filling in missing values of some columns, using other columns as anchors.

TUS does not guarantee to return such information. Many webtables in the dataset may not provide coverage over all query columns of Q. Consequently,

when many webtables have highly unionable alignments with a specific subset S of query columns with no unionable pair over the rest of columns, $Q-S$, TUS will return the top-k of them as the ranked list of webtables. In this scenario, the user has no information on the columns in $Q-S$ which makes it impossible to do follow up operations on these columns. For instance, in our running example, none of the top-2 candidate webtables of TUS offers a unionable pair for columns q_3 and q_2. The necessary information is not available for the user to fill in the missing value of q_3 (Table 3.5). This highlights the importance of a preference that guarantees to offer at least one unionable webtable with a coverage over each query column. A more general preference should return webtables with the highest unionability over each subset of query columns which also guarantees the coverage of each single query column.

A naive approach is to go over all the candidate webtables in the dataset, find their alignments with Q, compute their TUScores, rank them and return the top-k candidates with the highest score. This approach is not efficient and scalable for large datasets since checking all the webtables is a time-consuming task which requires significant resources. The question is how we can retrieve these webtables efficiently without iterating over the whole dataset.

An Efficient Approach

Skyline is a well-known preference extensively studied in the literature [16], [32], [36]. It is commonly defined over a corpus of data points, each represented as a numeric vector, where one looks for those data points which are not dominated by others. A data point D_n represented by vector V_n dominates a data point D_m represented by vector V_m and shown as $D_n \succ D_m$ if it has as good as or better values over all dimensions and a better value over at least one dimension [16].

Since there is no monotone function for skyline, we cannot apply it as part of the WAND approach. But, if we manage to represent each candidate webtable as a numeric vector or a set of numeric vectors, we can utilize existing efficient algorithms for computing skyline to get the desired candidate webtables efficiently [4], [56], [57]. This is challenging as we don't want to lose

any important information in the transformation process. The first idea is to have, for each candidate webtable C, a vector V of size equal to the number of query columns, with each element V_j representing the highest unionability of any unionable pair between a column in C and the query column q_j. This vector may represent alignments that are invalid for C as one candidate column may have the highest unionability with multiple query columns or vice versa and that is not permitted in our settings. We don't allow conflicts in term of mapping one column to multiple columns of the other table. So, this representation may lose information whether by introducing conflicts or missing valid alignments.

Another idea is to represent each alignment $a(C, Q)$ as a vector V_a of size equal to the number of query columns with each element showing the unionability score of a pair of columns. Following this idea, each candidate webtable with multiple alignments can be represented as a set of numeric vectors. An observation is that for one candidate webtable, some of the alignments are a subset of the others. An alignment $a_i(C, Q)$ is a subset of another alignment $a_j(C, Q)$ if a_j contains all the unionable pairs in a_i and more. We can safely remove the vector representations of all the alignments like a_i as their information is already been covered by another alignment, a_j. This will reduce the space of vectors for the skyline algorithm and increase the efficiency of the process.

We can show that by utilizing existing algorithms on skyline, we can return the candidate webtables with the best alignment over each subset of query columns (Lemma 1). In other words, skyline will return all the desired webtables and more. Consequently, we can employ current algorithms in the literature for skyline such as *SaLSa* [4], *BBS* [57] or *NN* [56] to find the webtables with the best alignments.

Lemma 1. With each alignment between a candidate webtable C and a query table Q represented as a numeric vector (as discussed), skyline preference over those vectors returns all the alignments with the highest AUScore and consequently their corresponding webtables with the highest TUScore.

Proof. We prove this by contradiction. Suppose the statement is false and there exist some webtable C_i with an alignment $a_i(C_i, Q)$ represented by vector V_i which has the highest AUScore over subset S of query columns but is not a member of skyline.

$$\begin{aligned} C_i \notin Skyline &\Rightarrow \exists\, C_l \in \mathcal{D}\ s.t.\ C_l \succ_S C_i \\ &\Rightarrow \forall\, q_j \in S \mid V_l[j] \ > \ V_i[j] \\ &\Rightarrow \prod_{q_j \in S} V_l[j] > \prod_{q_j \in S} V_i[j] \\ &\Rightarrow AUScore(a_l(C_l,\, Q)) > AUScore(a_i(C_i,\, Q)) \end{aligned}$$

This means there is another webtable C_l with a higher AUScore than C_i over S. This is a contradiction, hence the statement of the lemma is proved. □

Each webtable returned as part of the skyline results may have zero or more skyline vectors. A webtable with more skyline vectors is more likely to be unionable with the query table and maybe more interesting to be returned. After getting all the skyline vectors, we sort the corresponding webtables by how many skyline vectors they offer, with the webtable that offers more vectors being listed higher. Following this idea, we define top-k skyline to return the top k webtables of this list (Definition 3.8).

Definition 3.8 (Top-k Skyline)**.** With each alignment between a candidate webtable C and a query table Q represented as a numeric vector (as discussed), the *Top-k Skyline* of Q is the top-k candidate webtables with the most number of skyline vectors.

Applying this preference on samples presented in Table 3.2 gives us the skylines shown in Table 3.6. It is easy to show that for each subset of query columns, the best available webtables are part of skyline. For example, the best webtable for subset $\{q_1, q_2\}$ is C_2 with $TUScore = 0.786$ and for subset $\{q_3\}$ is C_1 with $TUScore = 0.767$ which both are part of skyline's results. To return the top-2 results, since all the four chosen candidates have the same number

CT	Vector Representation	Skyline
C_1	(c_{11} : 0.5, c_{12} : 0.2, c_{13} : 0.4, 0.0)	Yes
C_2	(c_{21} : 1.0, c_{22} : 0.4, 0.0, : 0.0)	Yes
C_3	(c_{31} : 0.5, c_{32} : 0.4, 0.0, : 0.0)	No ($C_2 \succ C_3$)
C_4	(c_{41} : 0.667, c_{42} : 0.5, 0.0, : 0.0)	Yes
C_5	(0.0, 0.0, c_{53} : 0.4, : 0.0)	No ($C_2 \succ C_5$)
C_6	(0.0, c_{61} : 0.143, 0.0, : 0.0)	No ($C_1 \succ C_6$)
C_7	(0.0, 0.0, 0.0, : 0.0)	No ($C_1 \succ C_6$)
C_8	(c_{81} : 0.167, 0.0, 0.0, : 0.0)	No ($C_1 \succ C_6$)
C_9	(0.0, 0.0, 0.0, c_{91} : 0.125)	Yes

Table 3.6: Results of applying skyline on running example (Table 3.2).

of skyline vectors, to break the tie, we return candidate webtables with the highest TUScore which are C_2 with score 0.786 and alignment $\{c_{21} \rightarrow q_1, c_{22} \rightarrow q_2\}$ and C_4 with 0.759 and alignment $\{c_{41} \rightarrow q_1,\ c_{42} \rightarrow q_2\}$. Notice that C_9 is part of the top-4 skyline as it is the only candidate webtable with a unionable pair for q_4, but it will not be part of top-4 TUS as there are other candidate webtables with better unionabilities. This is the situation where the skyline performs better as it offers candidates for each subset of query columns.

3.4.2 Diversity

There may be more than one interpretation of a query table, some of which are more important than others [62], [92]. A query table with two columns, 'country' and 'city', may belong to domains like cities, tourist attractions, soccer teams, etc. Webtables from aforementioned domains may be unioned with the query table over both columns. Users may be dissatisfied if they receive only results relating to one interpretation or domain of the query. Even if the query table is clear and unambiguous, users may not find it desirable to see webtables with the same columns as the query table and no new columns to offer [92]. Consider columns 'movie' and 'actor' of the query table Q of our running example (Table 3.1). In order to be helpful and user-friendly for some follow up operations, the returned webtables should include more columns than just movie and actor, such as role (i.e. C_2), director (i.e. C_3), etc. TUS ranks C_2, C_4 and C_1 higher than others while C_3 is more interesting than C_4 by having the director column.

Diversity is a well-studied preference in the literature with the purpose of resolving ambiguity by returning diverse results. Many approaches have been introduced to return diverse results in different contexts, such as *Information Retrieval* [20], [79], [92] and *Databases* [24], [29], [70], [78]. The proposed approaches usually involve a scoring function [78] which tries to find as diverse results as possible while promoting their relevancy to the search query.

As part of this work, we apply Vieira et al.'s framework [78] for diversifying the results of a SQL query to the context of table union search over webtables. This framework maximizes two different scoring functions, referred to as *sim* and *div*. *sim* calculates how similar the results are to the query, while *div* calculates how diverse they are compared to each other. To adopt this framework to table union search, we need to update both scoring functions (Formula 3.4). We define two scores $Relevancy(Rel)$ and $Difference(Dif)$ equivalent to sim and dif respectively with important changes. Rel scores guarantee similarity between candidate webtables and query tables using the unionability notions, while Dif scores ensure the webtables represent different interpretations of the query table, either coming from different domains or providing new dimensions. Building on this framework, we define top-k diversity as the list R of k candidate webtables which maximizes this objective function (Definition 3.9).

Definition 3.9 (Top-k Diversity)**.** The *Top-k Diversity* of query table Q over subset S of query columns is the list R of webtables from dataset $\mathcal{D}$ which maximizes the following objective function and are sorted based on their TUS-core.

$$Diversity(Q, S, k) = \underset{R \subseteq \mathcal{D}, |R|=k}{\operatorname{argmax}} (k-1).(1-\lambda).Rel(Q, S, R) + \lambda.Dif(Q, R). \tag{3.4}$$

In this formulation, λ is the parameter that controls the contribution of Rel and Dif scores to the final score. The term $(k-1)$ is to make sure that both scores have the same scale and range. Based on this framework, the Rel

score is defined over the whole list R of webtables as follows.

$$Rel(Q, S, R) = \sum_{i=1}^{|R|} \delta_{rel}(Q, S, R_i), \tag{3.5}$$

where δ_{rel} is the relevance of a single webtable to the query table Q over subset S of query columns. For computing the relevance of two tables we use their table unionability score as defined below.

$$\delta_{rel}(Q, S, R_i) = TUScore_S^{|S|}(R_i, Q). \tag{3.6}$$

The *Dif* score concentrates on two aspects: i) how different the webtables in R are from each other and ii) how different they are from the query table. Since we want both of these differences to be large at the same time, we define *Dif* as the harmonic mean of the two, defined as follows:

$$Dif(Q, R) = 2 * Dif_c(R) * Dif_q(R, Q)/(Dif_c(R) + Dif_q(R, Q)), \tag{3.7}$$

where Dif_c and Dif_q are the differences of the list R of webtables from each other and from the query table respectively. Each one of the differences can be formulated as follows:

$$Dif_c(R) = \sum_{i=1}^{|R|} \sum_{j=1, j\neq i}^{|R|} \delta_{dif}(R_i, R_j), \tag{3.8}$$

$$Dif_q(R, Q) = (|R| - 1) * \sum_{i=1}^{|R|} \delta_{dif}(R_i, Q), \tag{3.9}$$

where δ_{dif} is the difference of two tables R_i and R_j from each other and is defined as the number of columns in R_i that cannot form a unionable pair with a column in R_j. To bring δ_{dif} in range [0, 1] we divide it by the maximum degree of all webtables in the dataset $\mathcal{D}$.

We employ a unionability score threshold of 0.01 to filter out candidate webtables that cannot form a unionable pair. This is motivated by two reasons. Firstly, examining all candidate webtables can be a time-consuming task, especially when most of them are likely irrelevant to the query table. Thus, filtering out those with low unionabilities can help improve efficiency, even if it may result in some false negatives. Secondly, if we don't filter out irrelevant

webtables, they may offer many incorrect new columns for the query table. By using a small threshold, we ensure that the remaining candidate webtables are not completely irrelevant and have a higher likelihood of containing useful information.

$$\delta_{dif}(R_i, R_j) = \\ |\{c_i \in R_i \mid \nexists\, c_j \in R_j \; s.t. \; CUScore(c_i, c_j) > 0.01\}| \; / \underset{R_x \in \mathcal{D}}{\operatorname{argmax}} |degree(R_x)| \tag{3.10}$$

For finding the list R, a naive approach is to check every combination of webtables in the dataset $\mathcal{D}$ until the diversity score is the highest. But, this approach is not efficient or scalable and it is considered as a computationally hard task [78]. In order to compute a reasonable list R of webtables in time and memory, we need an efficient algorithm.

It is not possible to apply diversity to the WAND approach for two reasons. Firstly, there is no monotone function available for diversity. Secondly, the relationships between candidate webtables required for answering diversity are not preserved in the WAND approach. However, as we have a filtering and sorting of candidate webtables at the start of diversity, we can utilize WAND to get a sorted list of candidate webtables with an alignment over the input subset of query columns to the diversity preference.

An Efficient Approach

Greedy approaches resolve computationally hard problems by balancing efficiency and effectiveness. Our greedy approach is based on Yu et al.'s *Swap* algorithm [84]. The *Swap* algorithm starts with a list of k webtables and iterates over the rest of webtables to see if swapping outsiders with insiders improves the diversity score. After examining all the webtables and performing swap operations, the final list will be returned.

We modified the *Swap* algorithm to choose the initial list more carefully and iterate over webtables in a specific order. Since the focus is on returning the most diverse list R of webtables over a subset S of query columns, we

Algorithm 2 Diversity_Greedy_Algorithm: returns diverse list R of k unionable webtables from dataset $\mathcal{D}$ over subset S of query columns in Q.

$R = []$: the ranked list of results
PL: posting lists of columns in S
FC: webtables exist in all PL lists, sorted by #new domains for Q
for all $C_x \in FC$ **do**
    if $|R| < k$ **then**
        add C_x to R
    else if swapping C_x improves $Diversity(Q, S, k)$ **then**
        apply swapping with the biggest improvement
    else
        skip C_x
    end if
end for
Sort R by $TUScore_S^{|S|}$ and return it as output

Step	Description
0	FC = [C_1, C_2, C_3, C_4]
1	Top-2 = {C_1, C_2} with $Diversity = 0.887$
2	Swap C_3 with C_2: $Diversity = 1.064$ Swap C_3 with C_1: $Diversity = 0.831$ Top-2 = {C_2, C_3} with $Diversity = 1.064$
3	Swap C_4 with C_2: $Diversity = 0.883$ Swap C_4 with C_3: $Diversity = 0.939$ Top-2 = {C_2, C_3} with $Diversity = 1.064$

Table 3.7: Results and steps of applying top-2 diversity with $\lambda = 0.5$ over subset $S = \{q_1, q_2\}$ on running example (Table 3.2).

first filter all the webtables that do not have an alignment over S. In order to perform the filtering process in a more efficient manner without having to check every candidate webtable, we employ the WAND technique, which was previously discussed in Section 3.2. We then sort the filtered webtables by the number of new domains they are offering to the query table Q, with the ones offering more domains ranked higher. If a column c_i in a candidate webtable C cannot be unioned with any column q_j in Q with a unionability score over a small threshold (e.g. 0.01), it will be considered as a new domain offered by C. The small level of unionability shows that the two columns are potentially from different domains, although it is not always the case. At the end, we select the top k webtables as R and subsequently iterate over the rest to determine which swap operation improves the diversity score the most (Algorithm 2).

We applied this preference on the samples provided in Table 3.2 and the results are shown in Table 3.7. At the first step, an ordered list of webtables that are unionable with both q_1 and q_2 is retrieved, i.e. $\{C_1, C_2, C_3, C_4\}$. Then the first two webtables of the list, $\{C_1, C_2\}$, are chosen for being in the top-2 results. Next, we check if swapping C_3 with any of the webtables currently in the top-2 list makes an improvement on the diversity score. The swapping with the best improvement is replacing C_1 with C_3. The same process is repeated for C_4, and the final list $\{C_2, C_3\}$ is deemed as the best top-2 diverse webtables over subset $\{q_1, q_2\}$ of query columns. Each of these two webtables offer one new domain to Q whereas the tables that are not selected have the same domains as Q.

3.4.3 Novelty

Based on the notions of unionability previously defined in this work, candidate webtables that are near-duplicate versions of the query table, get a higher rank as the most unionable ones. They are therefore unsuitable for users who want to do specific follow up operations like expanding the query table vertically by adding rows. In this case, returning webtables with lower levels of unionability may be a better strategy than returning near-duplicate ones. Another similar situation that is returning candidate webtables that are not near-duplicate versions of the query table, but are near-duplicate versions of each other. In other words, candidate webtables may offer the same new information to the query table. The user will probably get the same information if only one candidate webtable is returned from each group of near-duplicate candidate webtables. Taking the running example (Section 3.3), candidate webtable C_4 is a near-duplicate version of Q and has no new information to offer. In contrast, C_3 offers three new rows to Q and although it is not among the top-2 webtables of TUS, it is a better candidate than C_1 and C_4.

Novelty is a well-known preference which has been studied in the literature with the purpose of avoiding redundancy in the returned result [13], [20], [28], [78]. In the context of our work, we try to avoid redundant webtables and return those offering as much unique new values as possible over a subset of

query columns. It can be considered as a special case of diversity preference which aims at finding webtables with the most diverse values over a subset of query columns. Hence, we can use the same framework discussed for the diversity preference with some modifications in the proposed scores.

This framework proposes an objective function consisting of two scores, *Rel* and *New*. The first score, *Rel*, computes the relevancy of the selected candidate webtables to the query table. On the other hand, the *New* score computes the amount of new information that the selected candidate webtables bring to the query table as a whole. We define top-k novelty over this framework as the list R of k candidate webtables which maximizes this objective function (Definition 3.10).

Definition 3.10 (Top-k Novelty)**.** The *Top-k Novelty* of query table Q over subset S of query columns is the list R of webtables from dataset $\mathcal{D}$ which maximizes the following objective function and are sorted based on their TUScore.

$$Novelty(Q, S, k) = \underset{R \subseteq \mathcal{D}, |R|=k}{\text{argmax}} (k-1).(1-\lambda).Rel(Q, S, R) + \lambda.New(Q, R), \tag{3.11}$$

where the parameters λ and $(k-1)$ in this formulation control the scores' contribution and to bring them to the same scale. The *Rel* score uses TUScore to compute the relevancy of webtables in R to the query table with the same formulation as before (Formula 3.5). The *New* score computes the novelty of the webtables in R and is composed of two different parts: i) the new information that a candidate table brings to the query table (New_q), preventing the selection of near-duplicate versions of Q, and ii) the new information that they bring to each other (New_c), avoiding the selection of near-duplicate webtables. As before, we take the harmonic mean of these two scores to ensure they are both large and significant.

$$New(Q, R) = 2 * New_c(R) * New_q(R, Q)/(New_c(R) + New_q(R, Q)), \tag{3.12}$$

$$New_c(R) = \sum_{i=1}^{|R|} \sum_{j=1, j\neq i}^{|R|} \delta_{new}(R_i, R_j), \tag{3.13}$$

$$New_q(R, Q) = (|R| - 1) * [\sum_{i=1}^{|R|} \delta_{new}(R_i, Q)], \tag{3.14}$$

where δ_{new} measures the novelty that one webtable brings. Since the focus is on returning webtables with new combinations of values over subset S of query columns, the following formulation for δ_{new} is proposed.

$$\begin{aligned} &\delta_{new}(R_i, R_j) = \\ &\quad |rows_S(R_i) - rows_S(R_j)| / \operatorname*{argmax}_{R_x \in \mathcal{D}} |cardinality(R_x)|, \end{aligned} \tag{3.15}$$

where $rows_S(R_i)$ returns the number of unique rows of webtable R_i over the subset S of query columns. To bring δ_{new} in range [0, 1], we divide it by the maximum cardinality of all webtables in the dataset $\mathcal{D}$.

We eliminate potential webtables that don't have at least one unionable pair with a score above a low threshold, such as 0.01. This is done for two reasons. Firstly, it increases efficiency as we don't have to analyze all potential webtables, including those that are irrelevant to the query table. Secondly, it prevents a webtable that is completely irrelevant to the query table and lacks a unionability pair from being mistakenly identified as the candidate with the highest New score.

To determine which list R of webtables has the highest novelty score, the naive approach involves checking every combination of webtables with an alignment over S and choosing the one with the highest scores. But, this approach is not efficient or scalable as it is a computationally hard task [78]. We need an efficient algorithm to compute an optimal list R of webtables in time and memory,

Similar to diversity, the WAND approach cannot incorporate novelty for two main reasons. Firstly, there is no available monotone function that can be used to measure novelty within this approach. Secondly, the relationships between candidate webtables that are necessary for determining novelty are not preserved in the WAND approach. Therefore, it is not possible to apply

novelty to the WAND approach. Again, we can employ WAND approach to filter candidate webtables that do not cover the input subset of query columns and to sort them based on their TUScore.

An Efficient Approach

It has been shown that finding the exact list R which maximizes the objective function is computationally hard [78], hence greedy approaches have been in use. In Algorithm 3, we do a similar process as we did for diversity preference by first removing all the irrelevant webtables with no alignment over subset S of query columns. Then, we sort the filtered webtables based on the number of unique rows they offer over S. Finally, following the *Swap* algorithm [84], we move over the sorted list and at each step try to swap the an outsider webtable with an insider one if it makes an improvement to the novelty score. The exchange with the maximum improvement will be applied at each step.

Algorithm 3 Novelty_Greedy_Algorithm: returns novel list R of k unionable webtables from dataset $\mathcal{D}$ over subset S of query columns in Q.

```
R = []: the ranked list of results
PL: posting lists of columns in S
FC: webtables exist in all PL lists, sorted by #new rows over S
for all C_x ∈ FC do
    if |R| < k then
        add C_x to R
    else if swapping C_x improves Novelty(Q, S, k) then
        apply swapping with the biggest improvement
    else
        skip C_x
    end if
end for
Sort R by TUScore_S^|S| and return it as output
```

Table 3.8 shows the steps the algorithm takes to get to the top-2 results over subset $\{q_1, q_2\}$ of query columns. After getting the filtered list of webtables over the input subset of query columns, it tries to improve the novelty score by swapping an unexamined webtable with one of the currently chosen ones. At Step 3, webtable C_3 makes an improvement and takes C_1's place in the top-2 results. Moving forward, C_4 is unable to get to the top-2 list and the

Step	Description
0	FC = [C_1, C_2, C_3, C_4]
1	Top-2 = {C_1, C_2} with $Novelty = 1.465$
2	Swap C_3 with C_2: $Novelty = 1.818$ Swap C_3 with C_1: $Novelty = 1.275$ Top-2 = {C_2, C_3} with $Novelty = 1.818$
3	Swap C_4 with C_2: $Novelty = 1.193$ Swap C_4 with C_3: $Novelty = 1.398$ Top-2 = {C_2, C_3} with $Novelty = 1.818$

Table 3.8: Results and steps of applying top-2 novelty with $\lambda = 0.5$ over subset $S = \{q_1, q_2\}$ on running example (Table 3.2).

list $\{C_2, C_3\}$ of webtables will be returned as the best top-2 webtables for the novelty preference. C_2 and C_3 both have 3 unique rows for Q over $\{q_1, q_2\}$ which are the maximum among all five webtables.

3.4.4 Dependent Set

A problem with the notions of unionability used in TUS is that each query column is treated as an independent entity. Consequently, important information about the combination of values across query columns will be lost. Consider the query table Q of the running example (Table 3.1) with a focus on query columns 'movie' and 'actor'. If we concentrate on individual query columns, there may be candidate webtables with similar values (for example, same movies and same actors) like C_2 and C_4. However, as there is a many-to-many relationship between movies and actors, these candidate webtables offer different combination of values over the two columns. TUS, considers all these candidate webtables as the same, as far as they have the same set of movies and actors. But, the data of these two webtables show that C_4 is a near-duplicate version of Q while C_2, although the first two rows' values appear in Q's columns, offers one new combination of them.

In order to resolve this issue, we propose dependent set preference, which differentiates between candidate webtables that have the same values but different combinations. Dependent set introduces sets of dependent query columns. Users can put query columns in different dependent sets to demonstrate the importance of their values together. Note that some query columns

may not be part of any dependent set or treated as a dependent set of size one. With this preference, each dependent set S_i of query columns is considered as a new query column with values generated by concatenating the values of columns in S_i. Following this idea, we define top-k dependent set as the top k candidate webtables with the most table unionability score over the set of newly created query columns (Definition 3.11).

Definition 3.11 (Top-k Dependent Set)**.** Having sets $S_1, \ldots, S_m$ of dependent query columns, the *Top-k Dependent Set* is defined as top-k candidate webtables of applying top-k TUS over the new set of query columns $\{q'_1, \ldots, q'_m\}$ $\cup$ $\{q_i \mid q_i \notin S_j \; for \; j = 1, \ldots, m\}$ where q'_i's values are the concatenation of values of query columns in S_i.

A naive approach to find dependent sets is to concatenate different combinations of candidate columns in every possible order for every webtable in the dataset and then find the most unionable combination over the new query columns. Since each candidate webtable can have many combinations of columns, examining all of them is not efficient. In addition, the exponential growth in possible combinations due to the increase in the number of candidate query columns makes this less scalable. We need a more efficient approach to find the desired webtables.

An Efficient Approach

The main problem with the naive approach is the huge space of possible combinations of columns that exists for candidate webtables. Pruning this space is the strategy we follow for evaluating this preference efficiently. We first find webtables with at least one alignment over a dependent set of query columns using the WAND approach over query columns' posting lists. Then, for each alignment of the filtered webtables, if it covers dependent set S_i, we concatenate candidate columns in the same order we did for q'_i to get to a new candidate column c'_i. The last step is to compute the table unionability score over the new set of candidate and query columns and to return the top-k ones

to the user (Definition 3.11).

Algorithm 4 Dependent_Sets_Greedy_Algorithm: returns the top-k webtables from dataset $\mathcal{D}$ with the maximum TUScore over sets S_1, ..., S_m of dependent query columns.

$R = []$: the ranked list of results
PL: posting lists of all columns
FC: webtables exist in all PL lists of at least one S_i
for all $i \in [1, m]$ **do**
    $q'_i = concatenate(S_i)$ // new query column
end for
for all $C_x \in FC$ **do**
    $A'(C_x, Q)$: alignments in $A(C_x, Q)$ covering at least one S_i
    for all $a(C_x, Q) \in A'(C_x, Q)$ **do**
        for all covered S_i **do**
            S'_i: candidate columns paired with S_i
            $c'_i = concatenate(S'_i)$ // new candidate column
        end for
        $a'(C_x, Q)$: alignment $a(C_x, Q)$ over new columns
        $goodness = AGScore(a'(C_x, Q))$
        if $|R| < k$ **then**
            add $(C_x, a(C_x, Q))$ to R
        else if *goodness* > minimum score in R **then**
            swap webtable with minimum score with $(C_x, a(C_x, Q)))$
        else
            skip C_x
        end if
    end for
end for
return R it as output

The results of applying top-2 dependent set on samples of Table 3.2 for two dependent sets $S_1 = \{q_1, q_2\}$ and $S_2 = \{q_3\}$ are shown in Table 3.9. Since we are using *Jaccard* as our column unionability score, some candidate tables like C_2 gets a score of 0.0 because values of concatenation of corresponding candidate columns, $c_2 1$ and $c_2 2$, has no overlap with values of concatenation of query columns, q_1 and q_2. Finally, the top-2 dependent set returns C_5 and C_1 as top webtables.

FC = [C_1, C_2, C_3, C_4, C_5] $q'_1 = q_1.q_2$, $q'_2 = q_3$, $q'_3 = q_4$		
Subset	**Top-2 AUScores**	**Top-2 AGScores**
$\{q'_1\}$	$C_4 = 0.4$, $C_1 : 0.334$	$C_2 : 0.767$, $C_4 : 0.731$
$\{q'_2\}$	$C_1 : 0.4$, $C_5 : 0.4$	$C_1 : 0.767$, $C_5 : 0.767$
$\{q'_1, q'_2\}$	$C_1 : 0.167$	$C_1 : 0.641$
others	None	None

Table 3.9: Results of applying top-2 dependent set with $k = 2$ for dependent sets $S_1 = \{q_1, q_2\}$ and $S_2 = \{q_3\}$ on running example (Table 3.2). $q_1.q_2$ denotes the concatenation of two columns.

3.4.5 Other Preferences

We can imagine other useful preferences but since they are not challenging in terms of implementing them efficiently, we just introduce them briefly here without further evaluations.

Weighting

In real word, query columns may have different levels of importance and some of them may be more important than others. The user can assign weights to query columns indicating how important they are. Consequently, the search aim at finding candidate webtables more in favor of important query column (i.e. tables with more unionability over the important columns). For the implementation, these weights can be reflected on the unionability scoring functions easily.

Total/Partial Order

The user may prefer some query columns over the others but with no specific weight. In this case user can provide a set of preferred relationships indicating one query column is favoured over another. There could be an order of preference between all the query column (i.e. Total Order) or just a subset of them (i.e. Partial Order).

Existence Set

A user may wants to have unionable webtables with at least an alignment over a specific subset of query columns. These query columns may be the primary

key of the query table or any sensitive or important information the user needs to see in the results.

Chapter 4

Experimental Evaluation

To evaluate the proposed preferences, we identify some down-stream tasks where table union search can be useful. One important goal of TUS is to augment a query table with additional information from a table corpus. This translates into three different down-stream tasks in our study: adding new rows to the query table, adding new columns, and filling in missing values.

This chapter begins by introducing two datasets in Section 4.1, followed by a discussion of data annotation and how the ground truth is established in Section 4.2.1 and some scoring functions in Section 4.2.3. We then proceed to evaluate our preferences in Section 4.3 by comparing them with the two variations of table union search, referred to as TUS and TUS+, in terms of precision and time cost. Finally, we evaluate our preferences over three down-stream tasks to show how the preferences may help in those tasks.

4.1 Datasets

We have collected two datasets commonly used in the literature: (1) *Web Data Commons - English-Language Relational Web Tables 2015 (WDC)* dataset [48], and (2) *WikiTables* dataset [5]. We build our dataset of query and candidate webtables from these base tables by random selection of rows, random projection of columns and random masking of values. Applying these operations on a base table is expected to generate new tables that are unionable with the base table and with each other by the definition of unionability [77]. This provides an easy annotation process as we know for each query table,

those tables that are sampled from the same base table as the query table are unionable with the query table.

4.1.1 WDC 2015

We selected 50 webtable with at least 9 columns and 900 records from the *WDC 2015* [48] dataset. Then we checked them manually and pruned those with less than 5 text columns or with non-English content. Finally we ended up with 14 webtables as our base tables. We sampled 101 tables from each base table, as discussed above, and designated one table as our query table and the remaining 100 tables as candidate webtables. This gave us a total of 14 query tables and 1400 candidate webtables. For each query table, we selected 100 to 300 rows and 4 to 6 columns randomly from the base table. Usually, query tables are expected to have a small number of rows. However, we specifically chose larger tables in order to observe the varying levels of unionability present in the candidate webtables, and to gain a better understanding of how each preference operates. We wanted to have smaller candidate webtables in order to have different levels of unionability with the query tables. So, we did the same for candidate webtables but with 100 to 200 rows and 3 to 4 columns. For each query table, we manually masked some cell values in 30 percent of the rows but limited the number of masked value for each row to only one. We kept the original cell values to be able to test the task of finding missing values later in the evaluation phase.

4.1.2 WikiTables

We did a similar process as the one for the *WDC* dataset. We first selected top 100 webtables with at least 9 columns and 300 rows from the *WikiTables* dataset [5]. Then, we pruned those with non-English content. In addition, since we wanted to guarantee different levels of unionability among query and candidate columns, we pruned those tables with many columns that had only 'yes/no' values in their content. Finally we ended up with 12 base tables which we used to generate query and candidate webtables from. We ended up with 12 query tables, one per each base table, and 1200 candidate webtables, 100

candidates per each base table. We did our masking operation to add some missing values to random cells of the generated query tables, similar to the masking carried out over the *WDC* dataset.

4.2 Experimental Setup

4.2.1 Ground Truth

The way we generated our query and candidate tables from a base table allowed us to exploit the same base tables as the ground truth. For a query table taken from a base table, any row, column and cell value of the same base tables can be considered as expected results to a table union search under some preferences. For missing value prediction, since each query table is sampled from a subset of columns of a base table and this subset may not include a key of the base table, there can be multiple candidates for a missing value and they all may be treated as correct. We consider any hit on these values as a correct guess for that missing value.

There are two important points to consider. Firstly, as the query and candidate webtables are selected randomly, it is possible that some candidate webtables may not have anything in common with the query table that is drawn from the same base table. This can result in these candidates being identified as non-unionable for the query table since value overlap is used as the unionability scoring function. However, since all approaches in the evaluations use value overlap to identify unionable candidates, this situation does not affect the overall trends observed in the evaluations.

Secondly, since base tables may have similar values, candidate webtables drawn from one base table may be identified as unionable with query tables from other base tables. In such cases, we consider these candidate webtables as false positives in our evaluations. However, we still examine the information they provide for the query table during follow-up operations. If they offer new information that is present in the base tables but not the query table, we present them to the user. Otherwise, if there are inconsistencies between the information they provide and that of the base tables, we ignore them. Despite

this, this situation does not affect the trends seen in the evaluations.

4.2.2 Evaluation Metrics

First of all, we return a ranked list of top-k webtables for each preference and utilize *Average Precision@k* (Formula 4.2) to get a sense of the number of returned webtables in the top-k list that are truly unionable with the query table.

$$Precision@k = \frac{\#\text{ unionable webtables in the top-k list}}{k}, \tag{4.1}$$

$$Average\ Precision@k = \frac{\sum_{j=1}^{N} Precision@k\ for\ Q_j}{N}, \tag{4.2}$$

where N is the number of query tables we do the experiments on.

We also report the time cost which is the time it takes for our implementations of each preference to return the top-k webtables for one single query table.

Next, for each follow up task, we report *Average Precision@k*, *Average Recall@k* and *Average F1 Measure@k* to highlight the effectiveness of each of the proposed preferences. Note that since we have different follow up tasks, precision@k, recall@k and f1 measure@k should be updated accordingly. For example, for the task of adding new rows to the query table, they will be defined as follows:

$$Precision@k = \frac{\#\text{ correct new rows returned by webtables of top-k list}}{\#\text{ all new rows returned by webtables of top-k list}} \tag{4.3}$$

$$Recall@k = \frac{\#\text{ correct new rows returned by webtables of top-k list}}{\#\text{ all possible new rows in the base tables of query tables}} \tag{4.4}$$

$$F1\ Measure@k = \frac{2 * Precision@k * Recall@k}{Precision@k + Recall@k} \tag{4.5}$$

4.2.3 CUScore Functions

Earlier in the Section 3 we introduced CUScore of two columns as the Jaccard similarity of their sets of values. However, CUScore is not limited to Jaccard

and other functions may be suitable for more specific follow up operations. There are several different definitions of CUScore in Table 4.1 that we intend to experiment with in this research, as we believe they may be helpful for the follow-up tasks.

CUScore
$Jaccard(q, c) = \|V_q \cap V_c\|/\|V_q \cup V_c\|$ Jaccard similarity over all rows of Q
$Jaccard_{missingrows}(q, c) = \|V'_q \cap V_c\|/\|V'_q \cup V_c\|$ Jaccard similarity over rows of Q with missing values
$Similarity(q, c) = \|V_q \cap V_c\|/\|V_q\|$ Fraction of q covered by c
$Similarity\_Novelty(q, c) = HM(Jaccard(q, c), \|V_c - V_q\|/N)$ Similarity and novelty of c for q at the same time

Table 4.1: CUScore functions over a query column q from query table Q, with a set of values V_q, and a candidate column c from Table C, with a set of values V_c; V'_q is the set of values of columns q over only those rows of Q with at least one missing value; $\mathcal{D}$ is the set of all columns in the dataset; N is the normalization parameter which could be $\text{argmax}_{c \in \mathcal{D}} |V_c|$ and HM is the Harmonic Mean function.

An observation is that a user may search for candidate webtables with possible information about the missing values of the query table Q in order to fill them. Without loss of generality, we can assume that Q has two sets R_1 and R_2 of rows, where R_1 includes rows with a missing value on at least one of its columns and R_2 consists of those with no missing values at all. Using Jaccard may result in returning candidate columns, and hence candidate webtables, with similar information to rows in R_2. Such candidates may not cover rows of R_1 which results in lower chances of finding the missing values. So, if we focus on returning candidate webtables with similar information to R_1, we increase the odds of finding the missing values. Based on this observation we define $Jaccard_{missingrows}(q, c)$ as the Jaccard similarity over only rows in R_1.

Another situation with using Jaccard is that two candidate columns with the same number of common values with a query column will not get the same CUScore if their numbers of non-common values are different. The column with less number of differences will get a higher score as it has a lower denominator in the Jaccard formula. We define overlap similarity, $Similarity(q, c)$,

to resolve this issue by focusing on just candidate column values that are common with the query.

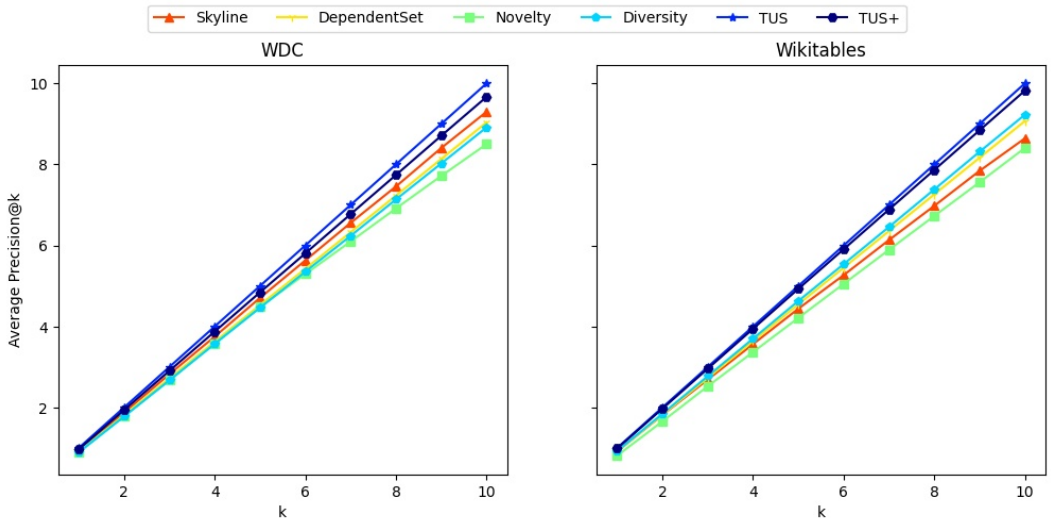
Similarity of two columns is only one measure of relatedness. Consider a candidate column with the exact same values as a query column. Jaccard will assign a unionability score of 1.0 to this candidate column. For a user who wants to see new values in candidate columns, or rows that are not in the query table, this column will not be desirable. This is where the differences become as important as similarities. We define $Similarity\_Novelty(q, c)$ as the harmonic mean of similarity and difference (i.e. novelty) of a candidate column to a query column.

For the evaluation, we experiment with all these $CuScore$ functions.

4.3 Experimental Results

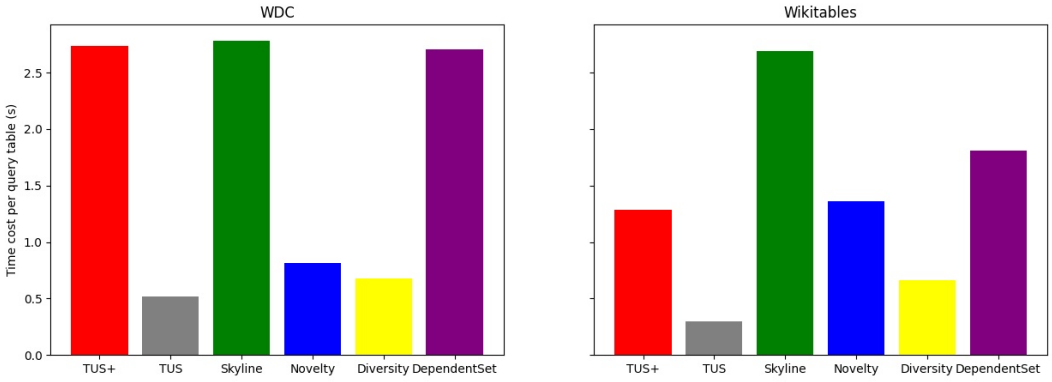
Since all the webtables from the same base table are considered as unionable, to evaluate the average precision@k of our proposed preferences, we run each preference over the datasets and measure the number of returned webtables that are drawn from the same base table as the query table. One important observation is that we consider, as unionable, only those webtables from the same base table as the query table, based on how our ground truth is constructed, whereas our definition of unionability allows webtables from other base tables to be considered unionable with the query table, resulting in lower levels of precision.

Figure 4.1: Average Precision@k for the returned webtables of both datasets.



As it is shown in Figure 4.1, the results of the proposed preferences is mostly made up of unionable webtables. Novelty seems to have a few non-unionable webtables as results which is expected since this preference cares about the new entities in the webtables as much as their relevancy to the query table. This may result in returning webtables from other base tables as unionable webtables for a query table. Another observation is that for smaller values of K, all the preferences have a perfect accuracy in returning only unionable candidate webtables.

Figure 4.2: Time cost of returning the ranked list of candidate webtables for a single query table.



For the evaluation of time cost, we compare the time it takes for each preference to find the desired webtables for a single query table with the times of TUS and TUS+. Since our preference performs extra processing over the candidate webtables, we expect them to take more time than TUS to return the results.

Plots in Figure 4.2 confirm our expectation that TUS has the lowest time cost among all the preferences. Skyline has the worst time cost which might be due to the recursive algorithm we used for its implementation. Existing approaches such as *SaLSa* [4] and *BBS* [57] should have a better performance than our approach as they utilize better heuristics to prune many candidate webtables without fully examining them.

4.3.1 Task 1: Adding New Rows

Consider the task of utilizing the candidate webtables that are unionable with the query table Q to extend Q vertically by adding new rows. The user may intend to train a machine learning model over the data in Q, for instance. Training a machine learning model on a table of data with a small number of rows, or a small sample size, presents several challenges [33] that can affect the model's accuracy and reliability. One of the most significant challenges is overfitting [59], where the model fits the training data too closely and fails to generalize to new data. With few observations, the sample might not be representative of the population, which can lead to biased training data and inaccurate predictions [44], [50]. Feature selection and model selection become more critical with a small sample size, as selecting irrelevant or redundant features can lead to overfitting, and choosing the appropriate model architecture can be more challenging [31]. Next we discuss some scenarios where TUS will be unable to return webtables suitable for such purpose while our proposed preferences may do better.

- The webtables retrieved from the search may only provide alignments that cover a specific subset of the query columns, leaving some columns without coverage. As a result, the user may not be able to construct

complete new rows for the query table due to insufficient information for the uncovered query columns.

- Although the returned webtables cover all the query columns, they may be divided into several groups, each offering an alignment over a different subset of query columns. Consequently, there is no common query column among webtables of different groups, making it impossible for the user to create full new rows for the query table by merging the webtables together.
- The returned webtables may cover all query columns and also be joinable, but they may be near duplicate versions of the query table, leaving the user with insufficient new rows.

Our hypothesis is that employing skyline, novelty, and dependent set preferences can aid in returning valuable candidate webtables in the aforementioned scenarios. Skyline ensures that each subset of query columns, as well as each query column, will be encompassed by at least one candidate webtable. Novelty identifies a group of candidate webtables with the highest count of unique rows across any selected subset of query columns by removing near-duplicate tables. Dependent set ensures that the retrieved tables pertain to the same entities as the query table and increasing the likelihood of generating additional new rows.

Evaluation Setup

To assess this task, we apply each of the proposed preferences on both datasets and for each query table. We examine the ranked list of candidate webtables returned by each preference and search for webtables that have an alignment with Q covering all of its query columns. For each webtable C that meets this requirement, we retrieve all of its rows, excluding those that already exist in Q, and consider them as new rows for Q. It should be noted that we need to rearrange the columns of C prior to returning its rows in order to maintain the mappings detected by the alignment between C and Q. Finally, any row provided by the returned webtables will be considered as a correct row if it

exists in the base table that Q was sampled from.

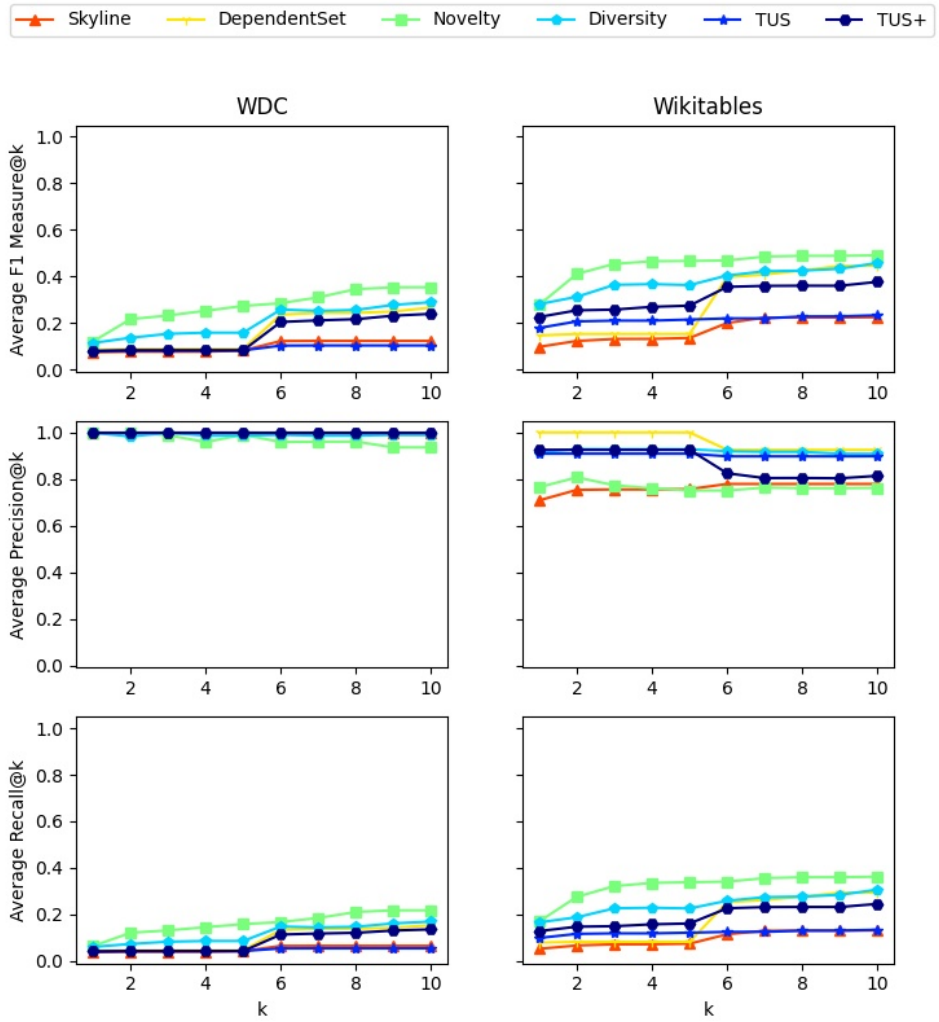
In accordance with the preference definitions outlined in Section 3.4, diversity, novelty and dependent set operate on a subset of query columns. As candidate webtables with an alignment over all the query columns are desired for this task, we execute these three preferences over the set of all query columns. Additionally, we set the controlling parameter λ to 0.5 for both diversity and novelty, indicating relevancy is as important as diversity and novelty scores.

Performance Evaluation

We first execute all the proposed preferences and both TUS implementations over the two datasets using the *Jaccard* scoring function. The results (Figure 4.3) show that novelty and diversity for all values of k and dependent set for $k > 5$ have a better f1 measure than that of TUS which means they offer more correct new rows to the query table. Novelty has the best performance as it focuses on returning as many new rows as possible. Interestingly, TUS+ which adds some simple constraints to TUS, as discussed in Section 3.2, makes a significant improvement on the result.

As expected, skyline has the worst performance among all the preferences, and clearly it is not a good preference for this task. The reason is that skyline is designed for situations where some query columns are rarely paired with candidate columns or the returned webtables need to be joined together to build full rows.

Figure 4.3: Task 1: Average F1 Measure@k, Average Precision@k and Average Recall@k of all approaches over both datasets.



Another observation is that the precision of all the proposed preferences is high which indicates that the returned webtables have indeed many correct new rows for Q. However, as it is shown, the recall is low, indicating that only a small portion of possible new rows have been found. There are multiple reasons for this. First, we only have a small number of candidate webtables for each base table which leaves many of the possible new rows uncovered by available candidate webtables. Second, for small values of k, we cannot achieve a high recall as each candidate webtable only covers a small portion of the base

table.

Changing the CUScore function

Now that we showed most of the proposed preferences outperform TUS and TUS+ for this task, we want to experiment if changing the CUScore function have an effect on the results. We first evaluate TUS and TUS+ using different CUScore functions to find out which one yields the best f1 measure. The results (Figures 4.4 and 4.5) show that both implementations, TUS and TUS+, works slightly better under *Similarity* scoring function. Notice that since $Jaccard_{missingrows}$ is designed for filling in the missing values task, it is not part of this experiment.

Figure 4.4: Task 1: Average F1 Measure@k for TUS on different CUScore functions over both datasets.

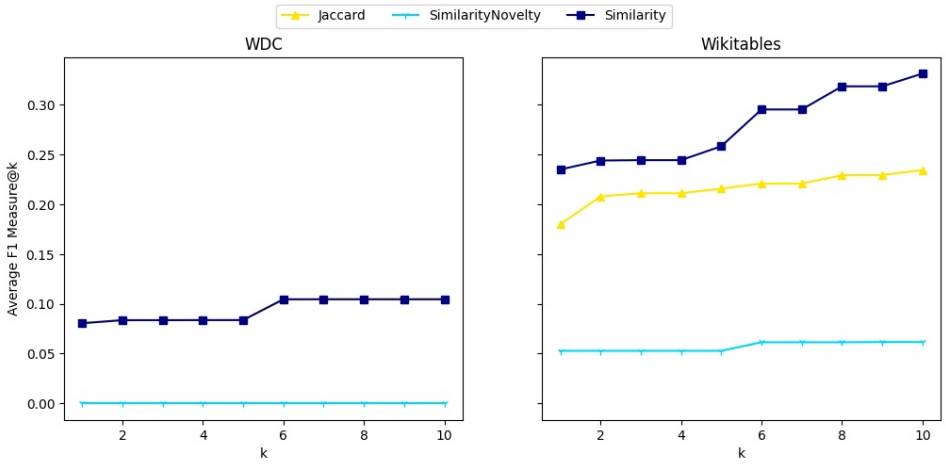
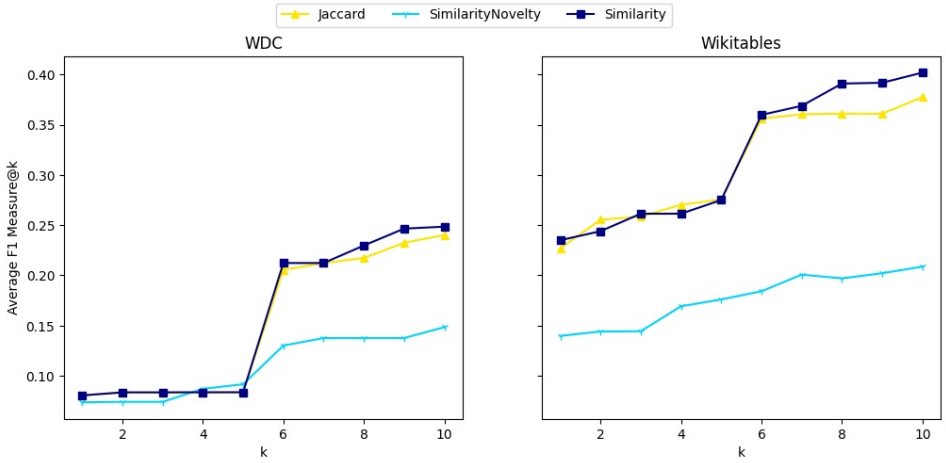
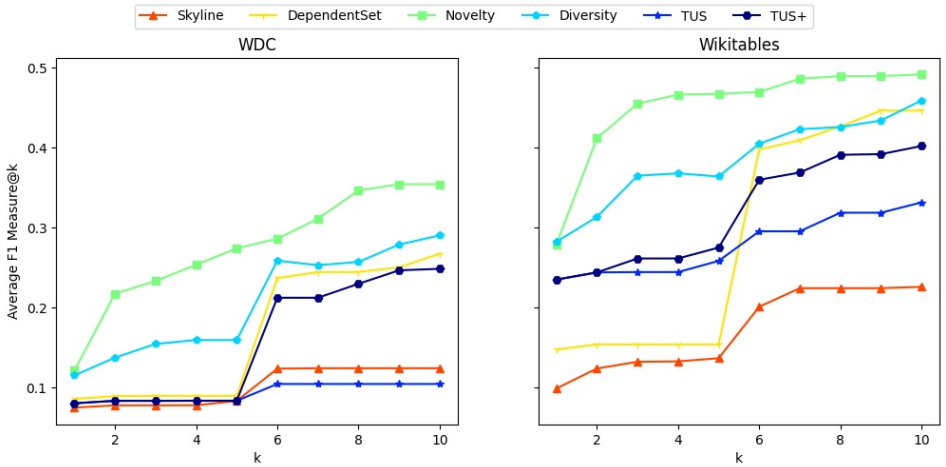


Figure 4.5: Task 1: Average F1 Measure@k for TUS+ on different CUScore functions over both datasets.



Comparing TUS and TUS+ using their best CUScore function with the proposed preferences on the *Jaccard* scoring function (Figure 4.6) shows the same trends we saw in Figure 4.3 and demonstrates that our preferences are still better than changing the scoring function.

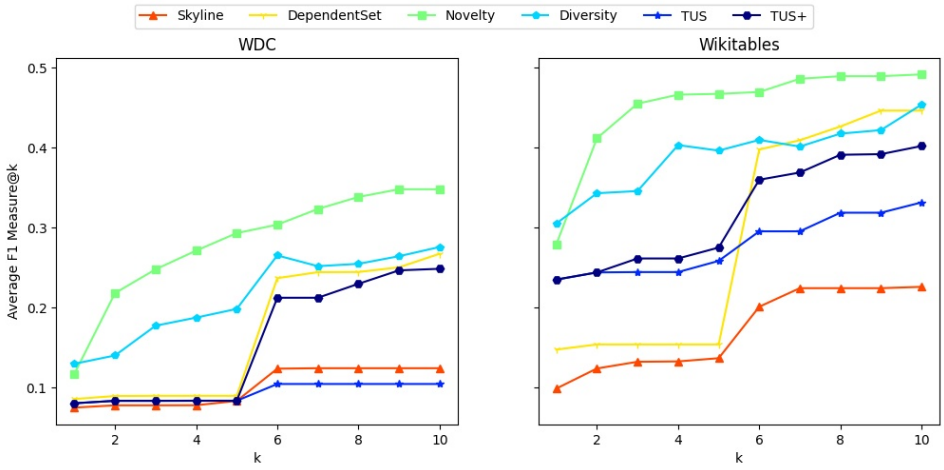
Figure 4.6: Task 1: Average F1 Measure@k for TUS and TUS+ with their best CUScore function and all preferences with Jaccard as CUScore function over both datasets.



The question now is how these CUScore functions can impact our preferences. To answer this question, we evaluate each preference using different CUScore functions across both datasets. Our experiments show that the

Jaccard scoring function is the best CUScore function for skyline and dependent set, while the *Similarity* function is the best for others. Figure 4.7 shows all the preferences and TUS implementations with their best CUScore function, which exhibits a similar pattern to Figure 4.6, confirming that the CUScore functions have a negligible effect on some preferences for the task of adding new rows.

Figure 4.7: Task 1: Average F1 Measure@k for TUS, TUS+ and all preferences with their best CUScore functions over both datasets.

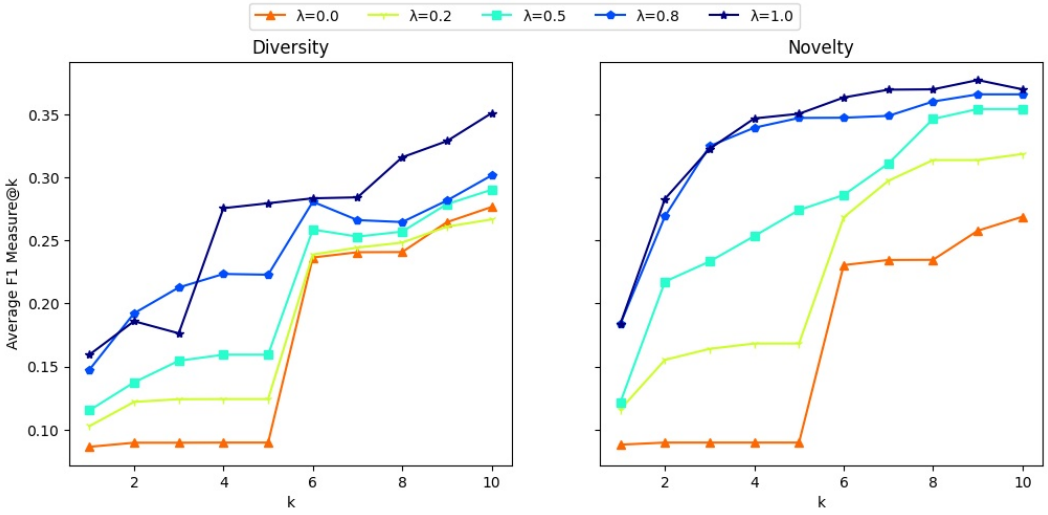


Changing the diversity/novelty parameter

As the previous evaluations have indicated that novelty and diversity are effective in expanding the query table vertically, we want to examine how changing their control parameter λ affects their performance. Based on the formulas for these preferences (Formulas 3.4 and 3.11), a value of $\lambda = 0.0$ corresponds to full relevance, which is equivalent to TUS. A value of $\lambda = 1.0$ indicates full diversity or novelty and could result in webtables with more new rows or columns. Figure 4.8 illustrates the results of applying both preferences to the *WDC* dataset for various λ values, namely $0.0, : 0.2, : 0.5, : 0.8, : 1.0$. We can conclude that for novelty, larger values of λ leads to more new rows being returned. This is due to two reasons. Firstly, we eliminate many irrelevant webtables at the beginning of both preferences by removing those candidate columns with a unionability score below a very low threshold. This helps to fil-

ter out webtables that were unlikely to be useful for the query table. Secondly, the base tables that were selected have a very low similarity in their values when compared to each other. As a result, there are no candidate webtables from other base tables that passes the threshold filter. Consequently, we end up with a long list of relevant candidate webtables to select the top-k from. The higher the value of λ, the more intriguing the webtable is for inclusion in the top-k list. The same trend is observed for diversity, although it focuses on returning webtables with more new columns for the query table. Upon examining the dataset, we discovered that webtables with more new rows are mostly those with new columns as well, which explains why we see similar patterns for diversity.

Figure 4.8: Task 1: Analyzing the effect of parameter λ on Average F1 Measure@k of diversity and novelty over *WDC* dataset.



4.3.2 Task 2: Adding New Columns

This task utilizes candidate webtables returned for a query to add new columns to the query table. The query table may have a few number of columns, making it challenging for a user intending to utilize Q for training a machine learning model, for instance. Training a machine learning model on a table of data with a low number of columns, also known as a low dimensionality dataset, presents several challenges that can affect the model's accuracy and reliability [37]. A

major challenge is underfitting, where the model might not have enough information to learn the underlying patterns and relationships in the data, leading to poor predictive performance [14], [55]. With few columns, there is limited information available to the model, which can lead to a loss of predictive power and accuracy. Furthermore, the limited number of features can lead to the inclusion of irrelevant or redundant features, reducing the model's predictive power and increasing the risk of overfitting [26]. The interpretability of the model's predictions can also be challenging with few columns, and there are limited options for feature engineering, which can limit the model's ability to learn complex relationships between features [19]. Here are a few cases where TUS may fail to return desirable candidate webtables for this task.

- Since the focus of TUS is on unionability, all returned candidate columns may be from the same domains as those of the query columns. Hence, no new domains may be offered by the returned webtables, which means no new dimensions can be added to the query table.
- The returned webtables may offer new columns to the query table, but they may have duplicate domains. As a result, the user ends up with a few new columns for the query table which may not be satisfactory.

In Section 3.4.2, we proposed diversity for the task of adding new columns to the query table. Our hypothesis is that this preference can retrieve better candidate tables for this task.

Evaluation Setup

One way to identify new columns for a query table Q is to obtain candidate webtables with an alignment that covers all the query columns. Any columns in these candidate webtables that do not participate in the alignment can be considered as new columns for Q. Another approach involves searching for candidate webtables with an alignment over a key of Q, which is a subset of columns that uniquely identifies a row of the table. As Q typically has a small cardinality, several column combinations may be identified as its keys. We select some of the largest keys (i.e., those with the most columns) and look for

candidate webtables with an alignment over each one of them. Any columns in the resulting candidate webtables that have no mapping with query columns may be potential new columns for Q. A new columns is a hit if it exists in the base table that Q is drawn from.

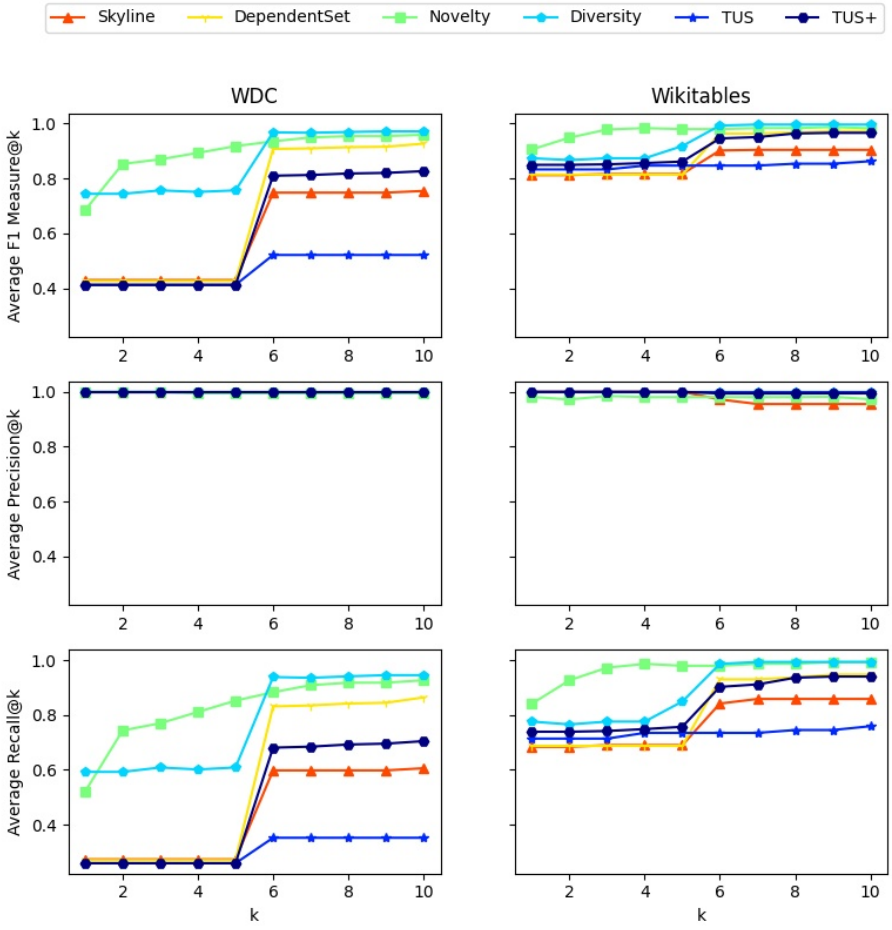
It's worth noting that diversity, novelty, and dependent set require an input set of query columns to function. Therefore, we first execute these metrics over all query columns, and then separately over some of the largest keys of Q. Additionally, we set the controlling parameter λ to 0.5 for both diversity and novelty.

Performance Evaluation

The first step is to execute all proposed preferences and TUS implementations over both datasets using the *Jaccard* scoring function. The average f1 measure results for various values of k indicate that among all the preferences, diversity performs the best, as expected, since its primary goal is to identify candidate webtables that have new columns for Q. Additionally, novelty is also effective in this task since webtables with more new columns also tend to have more new rows in our datasets. Dependent set puts the most desirable candidate webtables for this task in ranks 6 to 10 which is still better than that of TUS and TUS+. However, TUS exhibits the poorest performance, implying that its results may not be of high quality for this particular task. Interestingly, adding some constraints to TUS, denoted by TUS+, improves its results significantly. Nevertheless, our proposed preferences outperform TUS+ as well, indicating that we cannot achieve the same results as the preferences by implementing simple heuristics on TUS. Figure 4.9 displays the relevant plots.

A notable observation is that the average precision for nearly all the plots in both datasets reaches the peak, i.e. 1.0. This indicates that the candidate webtables returned are relevant to the query table and the new columns they provide are correct. Despite having a high average precision, the low average recall of some preferences indicates that their returned candidate webtables do not encompass all possible new columns of the base table for Q.

Figure 4.9: Task 2: Average F1 Measure@k, Average Precision@k and Average Recall@k of all approaches over both datasets.



Changing the CUScore function

In this section, we investigate whether altering the CUScore function affects the performance of TUS and TUS+. Our hypothesis is that while changing the scoring function may improve the performance, the preferences still yield better results. We use the same approach discussed in Section 4.3.1, starting with identifying the CUScore function with the best average f1 measure for both TUS and TUS+. Our evaluations, as depicted in Figures 4.4 and 4.5, demonstrate that both implementations perform better under the $SimilarityNovelty$ scoring function. It is worth noting that the $Jaccard_{missingrows}$ scoring function, which aims to aid in filling missing values, is not included in these eval-

uations.

Figure 4.10: Task 2: Average F1 Measure@k for TUS on different CUScore functions over both datasets.

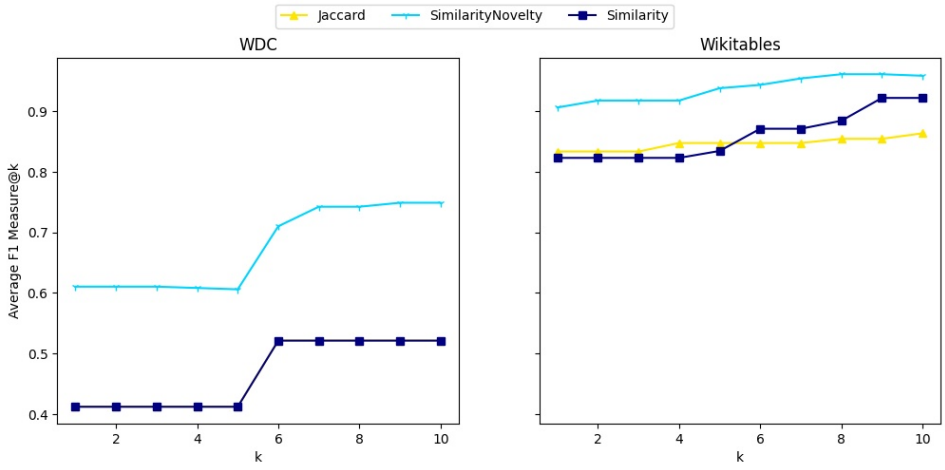
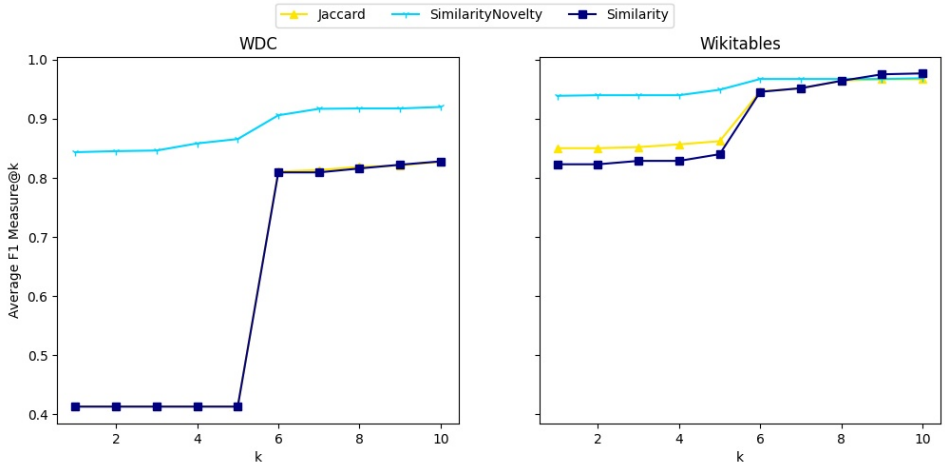
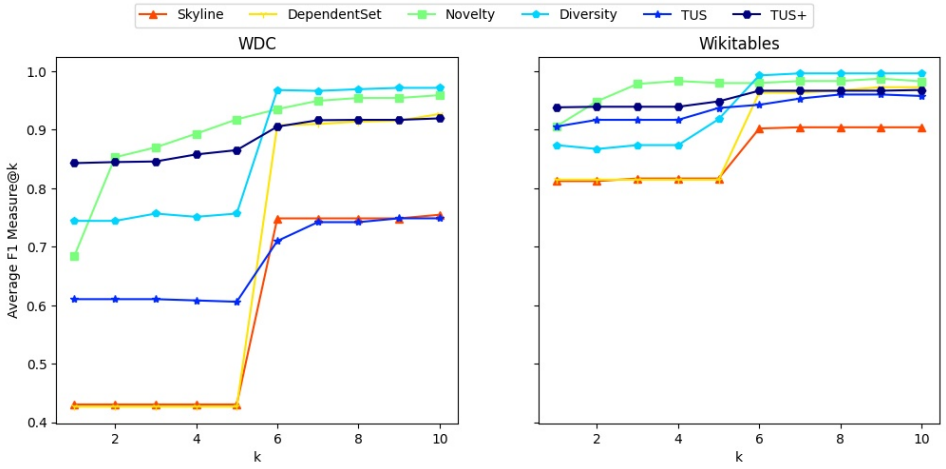


Figure 4.11: Task 2: Average F1 Measure@k for TUS+ on different CUScore functions over both datasets.



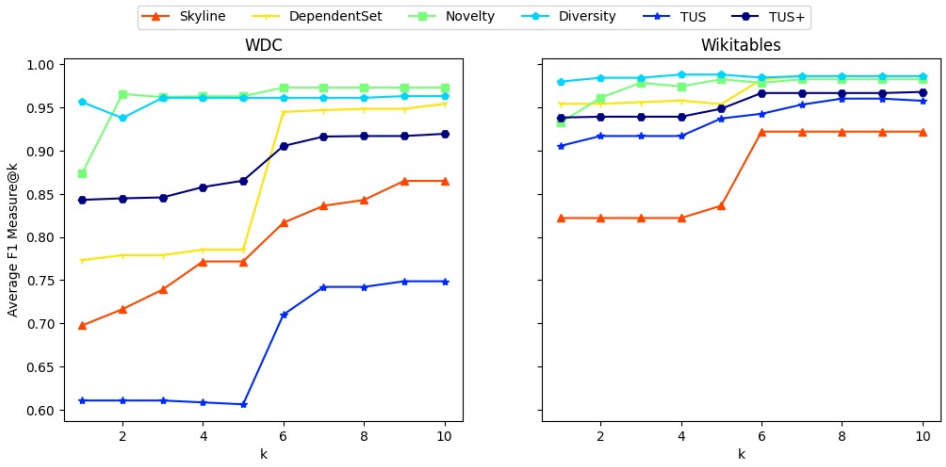
When comparing TUS and TUS+ using their best CUScore function with the proposed preferences using the *Jaccard* scoring function (as depicted in Figure 4.12), we observe similar trends as those in Figure 4.9. We notice that TUS+ outperforms skyline and dependent set preferences by a significant margin, but still lags behind diversity and novelty. Diversity and novelty are particularly effective for this task, particularly for relatively large values of k.

Figure 4.12: Task 2: Average F1 Measure@k for TUS and TUS+ with their best CUScore function and all preferences with Jaccard as CUScore function over both datasets.



Our goal is to understand the impact of CUScore functions on our preferences. To do so, we assess each preference using various CUScore functions on both datasets. The majority of preferences perform best with the CUScore function *SimilarityNovelty*, except for two instances where skyline and novelty perform better with *Similarity* on the Wikitables dataset. Figure 4.13 displays all preferences and both TUS implementations with their optimal CUScore functions, demonstrating similar patterns to Figure 4.12. Interestingly, the CUScore functions have only a small improvement on some preferences for the task of adding new columns. As shown, diversity and novelty have an average f1 measure over 0.95 and outperform others by a large margin for the WDC dataset and a small margin for the Wikitables dataset.

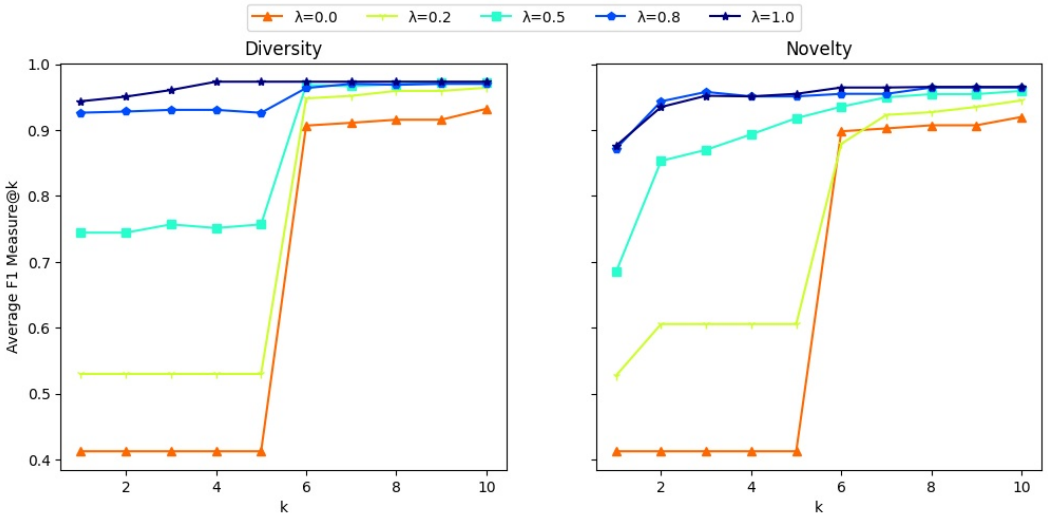
Figure 4.13: Task 2: Average F1 Measure@k for TUS, TUS+ and all preferences with their best CUScore functions over both datasets.



Changing the diversity/novelty parameter

We found that both novelty and diversity significantly outperform other preferences for adding new columns to extend the query table horizontally. Now, we want to explore how changing their control parameter λ affects their performance. As mentioned earlier, when $\lambda = 0.0$, these two preferences only consider the relevancy of returned webtables, whereas when $\lambda = 1.0$, they prioritize full novelty or diversity, which may result in returning webtables with more new columns or rows. Figure 4.14 illustrates the results of applying both preferences on the *WDC* dataset for various λ values, namely $0.0, : 0.2, : 0.5, : 0.8, : 1.0$. It's worth noting that for diversity, the larger the λ, the more new columns the preference offers. This pattern emerges from the fact that diversity initially filters many irrelevant webtables by pruning candidate columns with a unionability score below a small threshold. This leads to having a considerable number of relevant candidate webtables from which to choose the top-k. The more new columns a webtable has, i.e., the larger the λ, the more appealing it becomes to include it in the top-k list. We observe the same pattern for novelty, except that it focuses on returning webtables with more new rows for the query table. Our analysis of the dataset indicated that webtables with more new columns are usually the ones with new rows as well, which clarifies why we observe comparable trends for novelty.

Figure 4.14: Task 2: Analyzing the effect of parameter λ on Average F1 Measure@k of diversity and novelty over *WDC* dataset.



4.3.3 Task 3: Filling in Missing Values

The query table may have missing values. Consider a user who wants to train a machine learning model over the query table as the input dataset. Training a machine learning model on a dataset with missing values can present several challenges that can affect the accuracy of the model [80]. Missing values can cause biased training data, leading to inaccurate predictions. Handling missing values with imputation can be challenging, as selecting the appropriate imputation method is not always straightforward, and imputed values can introduce noise and inaccuracies [3]. Feature selection is also an essential consideration when dealing with missing values, as features with many missing values might not be useful predictors, and selecting relevant features can be challenging [35]. Here are a few cases where TUS can return candidate webtables not suitable for this task.

- Returned webtables may be all unionable over a specific subset of query columns which leaves some query columns uncovered. If any of the uncovered query columns contains missing values, the user will not be able to extract them.

- All the query columns with missing values may be covered by the returned webtables, but necessary information to extract the correct values for each missing value may not be accessible. We need a webtable with an alignment over all the query columns or with a primary key-foreign key relationship to the query table in order to extract the missing values from it. TUS does not guarantee that returned webtables meet these constraints.
- Even if the returned webtables address the first two situations, they may have near-duplicate information. Not having enough diverse values may result in not covering some of the missing values.

Skyline and dependent set preferences each addresses one or more of these situations and we expect them to be beneficial for this task. Skyline covers all the query columns and there is a chance of extracting missing values for each and every query column. We can also run novelty and dependent set over those query columns with missing values. They both return many combinations of values over those columns which helps in finding their missing values.

Evaluation Setup

To assess this task, preferences are applied and a list of candidate webtables is generated. From this list, the missing information in the query table Q is searched for. One approach to extract missing values is to find candidate webtables that have an alignment that covers all the columns in the query table. If there is a missing value at a particular column q in a specific row r of the query table, these webtables can be used to locate the row that is most similar to r. The value of the candidate column that is mapped to q can then be used as the potential value for the missing data.

Another strategy is to identify a primary key-foreign key relationship between the query table Q and a returned webtable. The approach is to first determine the key(s) of Q, and then check if any of them are covered by an alignment in a returned webtable. If such alignment(s) exist, they are treated as foreign keys to Q, and other candidate columns from the alignment are used

to fill in the missing values of their corresponding mapped query columns.

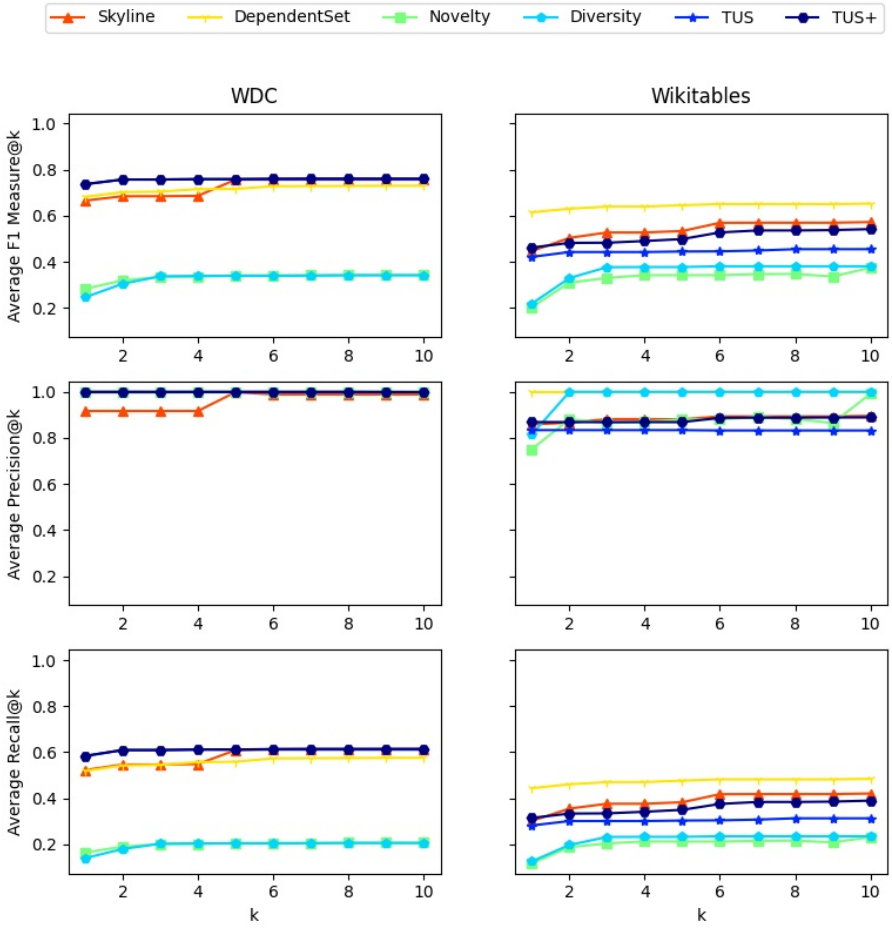
A different approach is to reverse the process of the previous strategy. The idea is to first identify the key(s) of any returned webtable C, and then verify if any of these keys are covered by an alignment. If such an alignment exists, the mapped columns of the query table are treated as foreign keys to C. This alignment can be utilized to extract missing values of Q.

The definitions in Section 3.4 highlights that to compute diversity, novelty, and dependent set preferences, an input set of query columns is required. Therefore, these preferences are initially executed over all the query columns and then over some of the largest keys of Q. The parameter λ is set to 0.5 for both diversity and novelty.

Performance Evaluation

As with other similar tasks, the initial step involves running all proposed preferences and TUS implementations over both datasets using the *Jaccard* scoring function. By varying the values of k, the average f1 measure results are obtained (Figure 4.15), and the results indicate that for the WDC dataset, the skyline and dependent set preferences perform well, meeting the expectation of high performance. However, TUS and TUS+ provide better results, which can be attributed to their focus on returning webtables with high unionability to Q, leading to more relevant webtables being returned, increasing the chances of finding missing values in Q. Dependent set and skyline show the best performance among all on the Wikitables dataset, which once again supports our hypothesis that these two would be beneficial in this task. Although diversity and novelty have high precision, they are not effective in detecting many of the missing values in Q, leading to a low f1 measure.

Figure 4.15: Task 3: Average F1 Measure@k, Average Precision@k and Average Recall@k of all approaches over both datasets.



Changing the CUScore function

This section examines how changing the CUScore function impacts the performance of TUS and TUS+. The first step is to determine the CUScore function that produces the highest average f1 measure for both TUS and TUS+. Figures 4.16 and 4.17 illustrate that both TUS and TUS+ perform better when using the *Similarity* scoring function on both datasets.

Figure 4.16: Task 3: Average F1 Measure@k for TUS on different CUScore functions over both datasets.

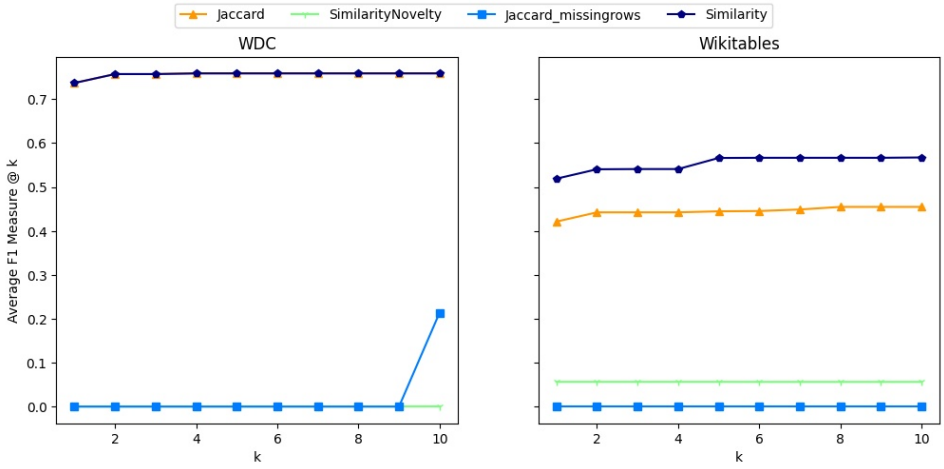
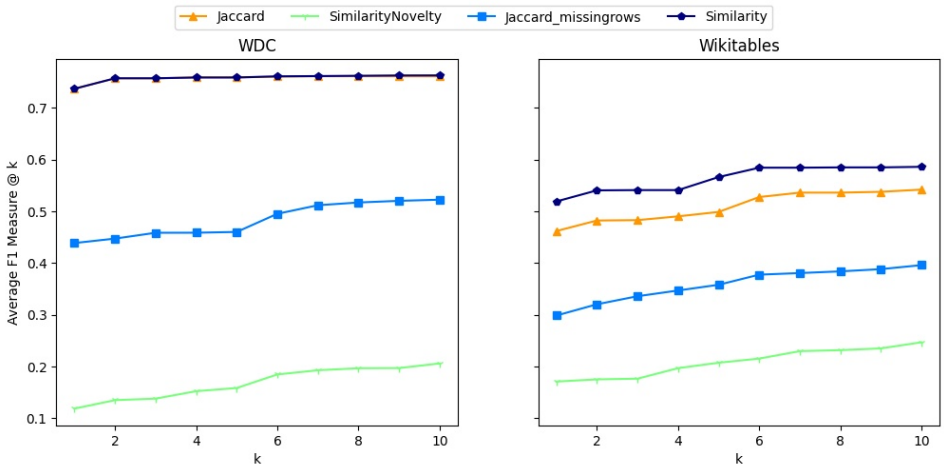
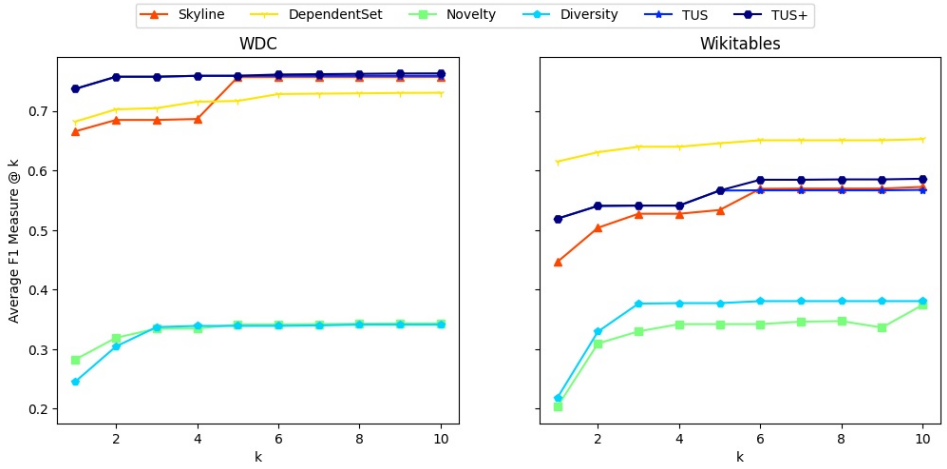


Figure 4.17: Task 3: Average F1 Measure@k for TUS+ on different CUScore functions over both datasets.



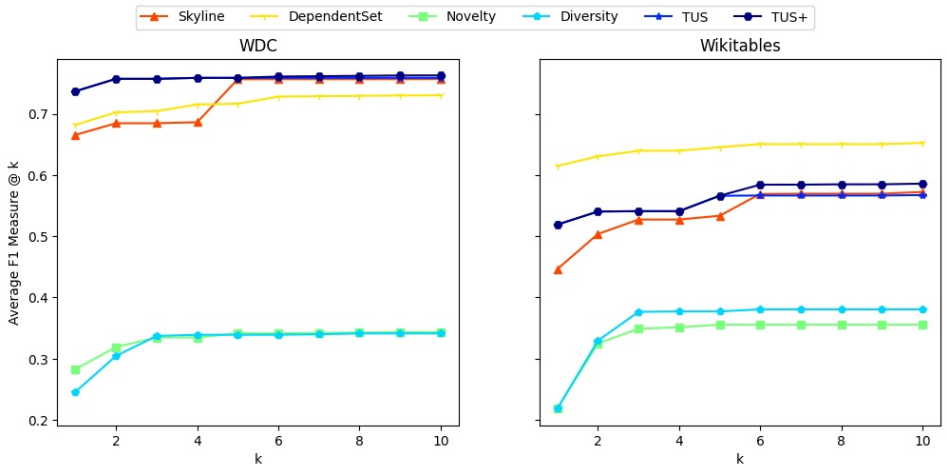
By comparing the performance of TUS and TUS+, using their best CUScore function, with the proposed preferences using the Jaccard scoring function, we find similar trends to those in Figure 4.15. However, for the Wikitables dataset, TUS and TUS+ perform slightly better than skyline. This observation is shown in Figure 4.18.

Figure 4.18: Task 3: Average F1 Measure@k for TUS and TUS+ with their best CUScore function and all preferences with Jaccard as CUScore function over both datasets.



The objective is to analyze the effect of CUScore functions on preferences, so we evaluate each preference using different CUScore functions on both datasets. Most preferences work better with the CUScore function *Jaccard*, except for novelty which performs slightly better with $Jaccard_{missingrows}$ on the Wikitables dataset. Figure 4.19 shows all preferences and both TUS implementations with their optimal CUScore functions, which indicates similar patterns to Figure 4.18. Interestingly, the new scoring functions do not enhance the majority of preferences in the task of filling in the missing values.

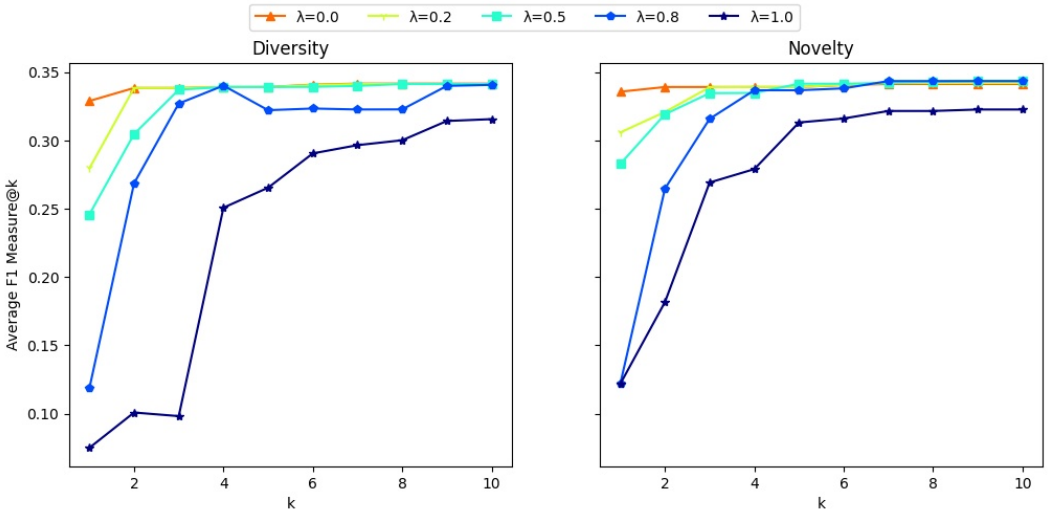
Figure 4.19: Task 3: Average F1 Measure@k for TUS, TUS+ and all preferences with their best CUScore functions over both datasets.



Changing the diversity/novelty parameter

While Novelty and Diversity were not effective in detecting missing values in the query table, we aimed to investigate how changing the λ parameter would impact their average f1 measure. The results, shown in Figure 4.20, indicate that decreasing the value of λ leads to higher average f1 measure for both preferences. This is because a lower λ value assigns a higher weight to the relevance of the candidate webtables to the query table, which increases the likelihood of finding the missing values.

Figure 4.20: Task 3: Analyzing the effect of parameter λ on Average F1 Measure@k of diversity and novelty over *WDC* dataset.



Chapter 5

Conclusion

Table Union Search aims at retrieving unionable webtables having a query table at hand. The literature mainly focus on efficiently returning an approximate ranked list of candidate webtables to the user. A drawback of this line of work is that this may prevent the user from accessing candidate webtables suitable for the follow-up operations, e.g. augmenting the query table with more rows and columns.

In this thesis, we study preferences as a powerful tool to improve the usage of TUS. Equipped with preferences, the user is able to retrieve webtables not only unionable with the query table, but also those suitable for various follow-up operations. We presented four main preferences *Skyline*, *Novelty*, *Diversity* and *Dependent Set* and showed that they can improve upon different down-stream tasks significantly.

Our evaluation on two real datasets showed that our proposed preferences offer significant improvements over three down-stream tasks studied in this thesis. Compared to both TUS algorithms, our preferences return more new rows, more new columns and more correct values for the gaps in the query table.

A possible future direction is to consider joining the returned webtables for down-stream tasks. Webtables have partial coverage over query columns which brings up the possibility of joining them in order to get to bigger webtables which consequently might add more rows and columns or fill more missing values of the query table.

Also, we can define and apply more preferences to TUS task in order to cover a wider range of down-stream tasks. In this work we only showed the benefits of a small set of preferences. Although these preferences can be used for other down-stream tasks, we only showed their usefulness on three down-stream tasks.

Another research direction is related to efficiency of proposed algorithms. In this research our focus was more on introducing and showing the effectiveness of preferences on TUS task. Although we proposed some efficient approaches for each preference, a possible direction is studying the efficiency of these algorithms and possibly developing more efficient solution, in terms of execution time and memory usage.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proceedings of the second ACM international conference on web search and data mining*, 2009, pp. 5–14.

[2] A. Ahmadov, M. Thiele, J. Eberius, W. Lehner, and R. Wrembel, "Towards a hybrid imputation approach using web tables," in *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, IEEE, 2015, pp. 21–30.

[3] M. Z. Asghar, M. Shahbaz, and F. S. Khan, "Missing data imputation using ensemble of machine learning algorithms," *Expert Systems with Applications*, vol. 128, pp. 70–87, 2019.

[4] I. Bartolini, P. Ciaccia, and M. Patella, "Salsa: Computing the skyline without scanning the whole sky," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, 2006, pp. 405–414.

[5] C. S. Bhagavatula, T. Noraset, and D. Downey, "Methods for exploring and mining tables on wikipedia," in *Proceedings of the ACM SIGKDD workshop on interactive data exploration and analytics*, 2013, pp. 18–26.

[6] C. S. Bhagavatula, T. Noraset, and D. Downey, "Tabel: Entity linking in web tables," in *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, Springer, 2015, pp. 425–441.

[7] A. Bogatu, A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, IEEE, 2020, pp. 709–720.

[8] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings 17th international conference on data engineering*, IEEE, 2001, pp. 421–430.

[9] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, "Efficient query evaluation using a two-level retrieval process," in *Proceedings of the twelfth international conference on Information and knowledge management*, 2003, pp. 426–434.

[10] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1090–1101, 2009.

[11] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: Exploring the power of tables on the web," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 538–549, 2008.

[12] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri, "Efficient diversification of web search results," *arXiv preprint arXiv:1105.4255*, 2011.

[13] P. Castells, N. Hurley, and S. Vargas, "Novelty and diversity in recommender systems," in *Recommender systems handbook*, Springer, 2022, pp. 603–646.

[14] G. Cawley and N. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2079–2107, 2016.

[15] Z. Chen, M. Trabelsi, J. Heflin, Y. Xu, and B. D. Davison, "Table search using a deep contextualized language model," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 589–598.

[16] J.-H. Choi, F. Hao, and A. Nasridinov, "Hi-sky: Hash index-based skyline query processing," *Applied Sciences*, vol. 10, no. 5, p. 1708, 2020.

[17] J. Chomicki, P. Ciaccia, and N. Meneghetti, "Skyline queries, front and back," *ACM SIGMOD Record*, vol. 42, no. 3, pp. 6–18, 2013.

[18] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, vol. 3, 2003, pp. 717–719.

[19] K. Chwialkowski and W. Jitkrittum, "Interpretability in low-dimensional models," in *Advances in Neural Information Processing Systems*, 2019, pp. 6654–6664.

[20] C. L. Clarke, M. Kolla, G. V. Cormack, *et al.*, "Novelty and diversity in information retrieval evaluation," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 659–666.

[21] T. Cong and H. Jagadish, "Pylon: Table union search through contrastive representation learning," *arXiv preprint arXiv:2301.04901*, 2023.

[22] A. Das Sarma, L. Fang, N. Gupta, *et al.*, "Finding related tables," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 817–828.

[23] T. Dasgupta and L. Dey, "Automatic scoring for innovativeness of textual ideas," in *Workshops at the thirtieth AAAI conference on artificial intelligence*, 2016.

[24] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl, "Divq: Diversification for keyword search over structured databases," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010, pp. 331–338.

[25] L. Deng, "Table2vec: Neural word and entity embeddings for table population and retrieval," M.S. thesis, University of Stavanger, Norway, 2018.

[26] Z. Farzaneh and H. Parvin, "Feature selection methods for high dimensional data: A review," *International Journal of Data Science and Analytics*, vol. 6, no. 4, pp. 277–289, 2018.

[27] R. C. Fernandez, E. Mansour, A. A. Qahtan, *et al.*, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, IEEE, 2018, pp. 989–1000.

[28] T. Ghosal, T. Saikh, T. Biswas, A. Ekbal, and P. Bhattacharyya, "Novelty detection: A perspective from natural language processing," *Computational Linguistics*, vol. 48, no. 1, pp. 77–117, 2022.

[29] O. Gkorgkas, A. Vlachou, C. Doulkeridis, and K. Nørvåg, "Finding the most diverse products using preference queries.," in *EDBT*, 2015, pp. 205–216.

[30] P. Godfrey, R. Shipley, J. Gryz, *et al.*, "Maximal vector computation in large data sets," in *VLDB*, vol. 5, 2005, pp. 229–240.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[32] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, "Scsa: Evaluating skyline queries in incomplete data," *Applied Intelligence*, vol. 49, no. 5, pp. 1636–1657, 2019.

[33] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[34] N. Hurley and M. Zhang, "Novelty and diversity in top-n recommendation–analysis and evaluation," *ACM Transactions on Internet Technology (TOIT)*, vol. 10, no. 4, pp. 1–30, 2011.

[35] X. Jiang, Y. Ouyang, and H. Liu, "A comprehensive study of feature selection methods for imbalanced data classification with missing values," *Expert Systems with Applications*, vol. 117, pp. 38–53, 2019.

[36] C. Kalyvas and T. Tzouramanis, "A survey of skyline query processing," *arXiv preprint arXiv:1704.01788*, 2017.

[37] A. Karami, F. Abas, and R. Darvishzadeh, "The impact of dimensionality reduction on text classification," *Journal of Information Science*, vol. 43, no. 1, pp. 85–96, 2017. DOI: 10.1177/0165551516658366.

[38] M. Karkali, F. Rousseau, A. Ntoulas, and M. Vazirgiannis, "Efficient online novelty detection in news streams," in *International conference on web information systems engineering*, Springer, 2013, pp. 57–71.

[39] Z. Kassenov, "Table expansion: Populating relational web tables based on examples," 2020.

[40] A. Khatiwada, G. Fan, R. Shraga, *et al.*, "Santos: Relationship-based semantic table union search," *arXiv preprint arXiv:2209.13589*, 2022.

[41] W. Kießling, "Preference queries with sv-semantics.," in *COMAD*, Citeseer, vol. 5, 2005, pp. 15–26.

[42] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, Elsevier, 2002, pp. 275–286.

[43] C. Koutras, G. Siachamis, A. Ionescu, *et al.*, "Valentine: Evaluating matching techniques for dataset discovery," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, IEEE, 2021, pp. 468–479.

[44] M. Kumar and D. Toshniwal, "Impact of sample size on machine learning model accuracy," *International Journal of Advanced Science and Technology*, vol. 29, no. 9, pp. 6983–6993, 2020.

[45] A. T. Kwee, F. S. Tsai, and W. Tang, "Sentence-level novelty detection in english and malay," in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2009, pp. 40–51.

[46] K. C. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-sky: An efficient skyline query processing framework based on z-order," *The VLDB Journal*, vol. 19, no. 3, pp. 333–362, 2010.

[47] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1502–1513, 2017.

[48] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 75–76.

[49] O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer, "The mannheim search join engine," *Journal of Web Semantics*, vol. 35, pp. 159–166, 2015.

[50] T. G. Lewis and D. B. Neill, "A survey of overfitting in decision tree learning," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3503–3539, 2016.

[51] R. J. Miller, "Open data integration," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2130–2139, 2018.

[52] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 813–825, 2018.

[53] A. Neelakantan, Q. V. Le, and I. Sutskever, "Neural programmer: Inducing latent programs with gradient descent," *arXiv preprint arXiv:1511.04834*, 2015.

[54] T. T. Nguyen, Q. V. H. Nguyen, M. Weidlich, and K. Aberer, "Result selection and summarization for web table search," in *2015 IEEE 31st International Conference on Data Engineering*, IEEE, 2015, pp. 231–242.

[55] I. Oseledets and D. Kovalev, "Deep learning for low-dimensional problems: A survey," *arXiv preprint arXiv:1711.07447*, 2017.

[56] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 467–478.

[57] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 41–82, 2005.

[58] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," *arXiv preprint arXiv:1508.00305*, 2015.

[59] M. A. Pimentel, D. A. Clifton, and L. Tarassenko, "Overcoming small sample size limitations in machine learning for medical applications," *Computerized Medical Imaging and Graphics*, vol. 38, no. 8, pp. 722–732, 2014.

[60] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *arXiv preprint arXiv:1207.0132*, 2012.

[61] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *arXiv preprint arXiv:1208.0076*, 2012.

[62] F. Radlinski and S. Dumais, "Improving personalized web search using result diversification," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006, pp. 691–692.

[63] D. Rafiei, K. Bharat, and A. Shukla, "Diversifying web search results," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 781–790.

[64] D. Ritze and C. Bizer, "Matching web tables to dbpedia-a feature utility study," *context*, vol. 42, no. 41, pp. 19–31, 2017.

[65] R. L. Santos, C. Macdonald, and I. Ounis, "Exploiting query reformulations for web search result diversification," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 881–890.

[66] S. Sarabchi, "Extending tables using a web table corpus," 2020.

[67] S. Sarawagi and S. Chakrabarti, "Open-domain quantity queries on web tables: Annotation, response, and consensus models," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 711–720.

[68] I. Soboroff, D. Harman, *et al.*, "Overview of the trec 2003 novelty track.," in *TREC*, 2003, pp. 38–53.

[69] I. Soboroff and D. Harman, "Novelty detection: The trec experience," in *Proceedings of human language technology conference and conference on empirical methods in natural language processing*, 2005, pp. 105–112.

[70] K. Stefanidis, M. Drosou, and E. Pitoura, "Perk: Personalized keyword search in relational databases through preferences," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 585–596.

[71] K. Stefanidis, G. Koutrika, and E. Pitoura, "A survey on representation, composition and application of preferences in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 36, no. 3, pp. 1–45, 2011.

[72] H. Sun, H. Ma, X. He, W.-t. Yih, Y. Su, and X. Yan, "Table cell search for question answering," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 771–782.

[73] K.-L. Tan, P.-K. Eng, B. C. Ooi, *et al.*, "Efficient progressive skyline computation," in *VLDB*, vol. 1, 2001, pp. 301–310.

[74] F. S. Tsai and Y. Zhang, "D2s: Document-to-sentence framework for novelty detection," *Knowledge and information systems*, vol. 29, no. 2, pp. 419–433, 2011.

[75] R. H. Van Leuken, L. Garcia, X. Olivares, and R. van Zwol, "Visual diversification of image search results," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 341–350.

[76] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. A. Yahia, "Efficient computation of diverse query results," in *2008 IEEE 24th International Conference on Data Engineering*, IEEE, 2008, pp. 228–236.

[77] P. Venetis, A. Y. Halevy, J. Madhavan, *et al.*, "Recovering semantics of tables on the web," 2011.

[78] M. R. Vieira, H. L. Razente, M. C. Barioni, *et al.*, "Divdb: A system for diversifying query results," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1395–1398, 2011.

[79] M. R. Vieira, H. L. Razente, M. C. Barioni, *et al.*, "On query result diversification," in *2011 IEEE 27th International Conference on Data Engineering*, IEEE, 2011, pp. 1163–1174.

[80] X. Wang, Z. Li, J. Wu, Y. Gao, and Y. Jiang, "A review of missing data treatment methods," *International Journal of Automation and Computing*, vol. 15, no. 6, pp. 643–655, 2018.

[81] D. Xin, H. Cheng, X. Yan, and J. Han, "Extracting redundancy-aware top-k patterns," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 444–453.

[82] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "Infogather: Entity augmentation and attribute discovery by holistic matching with web tables," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 97–108.

[83] Y. Yang, J. Zhang, J. Carbonell, and C. Jin, "Topic-conditioned novelty detection," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 688–693.

[84] C. Yu, L. Lakshmanan, and S. Amer-Yahia, "It takes variety to make a world: Diversification in recommender systems," in *Proceedings of the 12th international conference on extending database technology: Advances in database technology*, 2009, pp. 368–378.

[85] M. Zhang and K. Chakrabarti, "Infogather+ semantic matching and annotation of numeric and time-varying attributes in web tables," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 145–156.

[86] S. Zhang and K. Balog, "Entitables: Smart assistance for entity-focused tables," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 255–264.

[87] S. Zhang and K. Balog, "Ad hoc table retrieval using semantic similarity," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 1553–1562.

[88] S. Zhang and K. Balog, "Auto-completion for data cells in relational tables," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 761–770.

[89] S. Zhang and K. Balog, "Recommending related tables," *arXiv preprint arXiv:1907.03595*, 2019.

[90] X. Zhang, Y. Chen, J. Chen, X. Du, and L. Zou, "Mapping entity-attribute web tables to web-scale knowledge bases," in *International Conference on Database Systems for Advanced Applications*, Springer, 2013, pp. 108–122.

[91] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1951–1966.

[92] K. Zheng, H. Wang, Z. Qi, J. Li, and H. Gao, "A survey of query result diversification," *Knowledge and Information Systems*, vol. 51, no. 1, pp. 1–36, 2017.

[93] 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*, 2019, pp. 847–864.