ARTS: A System for Aggregate Related Table Search

Junjie Xing
Computer Science and Engineering
University of Michigan
Ann Arbor, USA
jjxing@umich.edu

H. V. Jagadish
Computer Science and Engineering
University of Michigan
Ann Arbor, USA
jag@umich.edu

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00428

***Abstract*—Existing table search techniques define table relatedness with unionablility and/or joinability. While these are valuable, they do not suffice for most data analysis tasks that involve numerical data, which is often aggregated over geographical, temporal, or other groups. In this demonstration, we showcase ARTS, a novel table search system centered on the unique concept of aggregate relatedness. By leveraging pre-trained language models, ARTS offers a superior column semantics understanding capability, with good labels created for both textual and numerical columns. This demonstration will offer attendees hands-on interaction with our system, revealing its potential in effectively addressing real-world data analysis challenges.**

***Index Terms*—table discovery, column semantics, data integration, large language model**

I. Introduction

Table search has been studied extensively in recent years. As ever more data becomes available on the web and in data repositories, and ever more applications are found for data analysis and machine learning, there is an ever greater demand to find data relevant to a task at hand. Consequently, the significance of table search continues to escalate.

Often, in table search scenarios, a user possesses an initial table and seeks supplementary tables, from a repository, or the web more broadly, to combine with their primary table. So they use their table as a query table to initiate the search. Several relatedness metrics have been developed by researchers to measure the association between a query table and potential matches. For instance, in the context of unionable table search problems, "unionability" is employed as a measure of table relatedness [1]–[4]. According to [1], two tables are considered unionable if they possess shared attributes from a common domain. However, if the user's purpose is quantitative data analytics, most attribute values of interest are numbers, often reported as aggregates rolled up at various granularities; and current relatedness metrics are not suitable. Not surprisingly, tables with aggregates are prevalent in open domain repositories, such as the US government's open data portal (Data.Gov) [5]. We've observed that over 50% of tables in Data.Gov comprise aggregated information.

In this demonstration, we showcase a system, ARTS, that uses a novel objective for table search, namely "aggregate relatedness". Two tables are deemed "*aggregate related*" if they can be connected via aggregate queries. Let's elucidate this concept with the following example:

Example 1 Consider Ela, an epidemiologist working for a city government, who has a COVID-19 table shown in Fig. 1 as the user's query table with aggregate information about positive cases, hospitalization, etc., by date, age group and sex. To compare the trend of daily COVID-19 cases against other jurisdictions, Ela aimed to find tables such as the sample output table in Fig. 1, which is another table, from a state government, with aggregated COVID-19 cases by date and sex. After applying an aggregate query on the first table, also shown in Fig. 1 as the aggregate transformation, the schema of the resulting table is equivalent with the second one. Note that not only are the dimension columns semantically matched, the numerical columns, "total" and "num_cases", also share the same semantics ("total number of COVID-19 cases"). In this case, the two tables are *aggregate related*.

The notion of "aggregate relatedness" posits a more stringent table relationship than mere table unionability. To classify two tables as "aggregate related", not only must the two tables be transformed with aggregation queries to tables on the same aggregation level so that the values can be compared, as illustrated in Example 1, but their columns should also share the same fine-grained column semantics. For example, a column about "number of students" should be distinguished from a column about "number of COVID-19 cases": their common column semantics type "number" is not sufficient.

Example 2 Using existing schema matching and table union search techniques, the table depicted in Figure 2 could be a perfect match with the query table in Figure 1, since the two tables share the similar table header, cell values, and even column data distribution. However, it makes no sense to union a table about autopsies performed in Chicago with the query table, which counts COVID-19 cases. Although table depicted in Figure 2 meets the requirement for being unionable with current table union search systems [1], [4], it falls short of our criteria for aggregate relatedness.

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00428

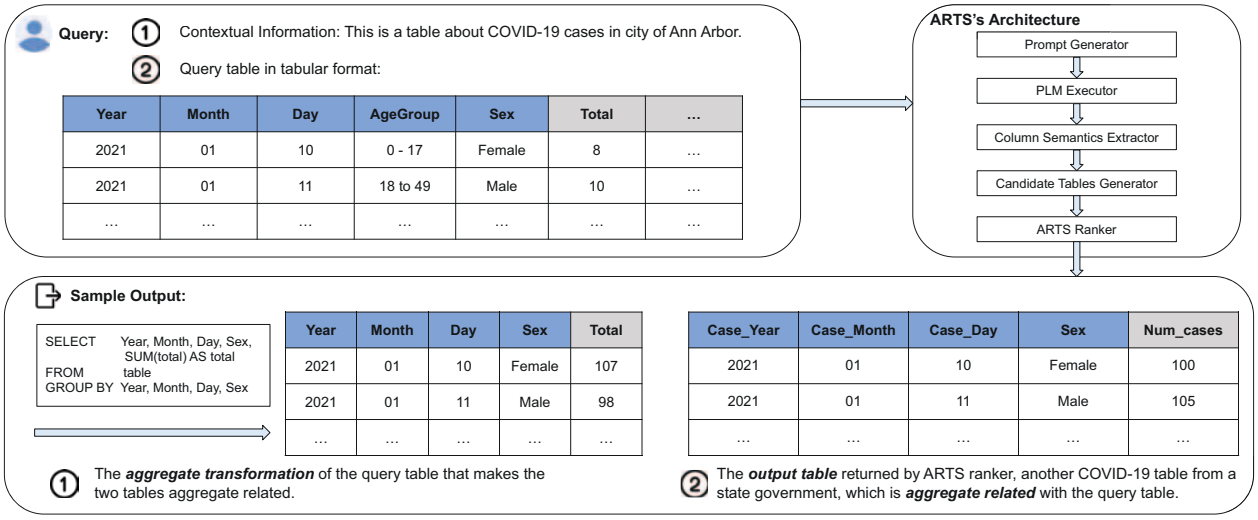

Fig. 1: ARTS's Architecture and Sample Input/Output

Year	Month	Day	Sex	Total
2020	05	10	Female	6
2020	05	10	Male	5
...	...	...	...	...

Fig. 2: A table about autopsies performed in Chicago.

In the following, we first introduce ARTS's framework in Sec. II, the exhibit the demonstration experience in Sec. III.

II. ARTS's Framework

This section presents the overall framework of the ARTS system. We discuss the user input, system output, ARTS's architecture, and ARTS's implementation detail.

A. User Input

We require the user to provide two things to form a query, as depicted in Fig. 1. First, the user should upload a query table in tabular format. Second, the user needs to provide required contextual information about the uploaded table, including the title of the table and a description about the table. ARTS will fill in the title of the table with the submitted tabular filename, however, the user has the right to modify it and provide a more accurate table title.

The effectiveness of ARTS relies on the accuracy of the contextual information that the user provided with the title and description. The contextual information will be fed into the next steps for a comprehensive table understanding of the query table, and assist the finding of aggregate related tables.

B. System Output

The system returns a list of output tables that are aggregated related with the query table, along with the aggregate transformations that makes the two tables aggregate related. The sample out in Fig. 1 illustrates a top ranked aggregate related table w.r.t. the query table in our table repository, and the aggregation transformation of the query table that makes the two tables aggregate related.

It is worth noting that not only the query table, but also the output table can be associated with an aggregate transformation if necessary. For example, if the sample system output table in Fig 1 is not aggregated by "case_year", "case_month", "case_day" and "sex", instead aggregated on one more column named "race/ethnicity", then the sample output table should be associated with an aggregate transformation on "case_year", "case_month", "case_day" and "sex".

C. ARTS's Architecture

As depicted in Fig. 1, ARTS consists of five modules, including a prompt generator, a pre-trained language model (PLM) executor, a column semantics extractor, a candidate table generator, and an aggregate related table ranker. In the following, we will introduce each modules in order. ARTS's success relies on the holistic understanding of fine-grained column semantics on the query table. We show how ARTS achieve this with the prompt generator, the PLM executor, and the column semantics extractor.

1) Prompt Generator: We use a template-based prompt generator that takes required contextual information of the table, and generates a prompt that asks for column semantics annotation from a pre-trained language model.

2) PLM Executor: A PLM executor takes a prompt as input, and a selected PLM as a target, and calls the API of the PLM to initiate a sequence completion task or a chat completion task. We will discuss the implementation of the PLM executor and the selection of PLMs in Sec. II-D.

An example of prompt and the response from PLM is illustrated in Figure 4, the PLM used is GPT-3.5-Turbo [6].

3) Column Semantics Extractor: A column semantics extractor takes the response from the PLM as input, and extracts the column semantics annotation for each column in the table.

For example, the extractor extracts "The year in which the COVID-19 cases were reported" for column "case_year".

It is worth highlighting that the success of the column semantics extractor relies on the ability of the PLM in adhering to provided instructions (in the prompt) and subsequently producing well-structured outputs.

4) Candidate Table Generator: For two tables to be aggregate related, they must share columns with the same fine-grained column semantics. Using this requirement, we generate a list of candidate tables by gathering the tables in the table repository that shares at least one common column with the column semantics extracted for the query table.

5) ARTS Ranker: The aggregate related table search ranker implements an aggregate relatedness score function. The score function takes into account the column semantics similarity between the two tables and the pairs of aggregate transformations that makes the two tables aggregate related.

D. ARTS's Implementation

As a table search system, we need to build a table repository as the search space. Furthermore, we need to pre-annotate the column semantics for all the tables in the table repository for an effective aggregate related table search.

To build a large table repository, it is impossible to manually provide an accurate title and description for all the tables. Instead, we used the U.S. Government's Open Data (Data.Gov) [5] as the data source. This choice was based on several reasons. Firstly, it is a large data portal, comprising over 250,000 datasets. Secondly, it adheres to the metadata standard DCAT-US Schema v1.1, which enforces metadata availability, including the title and description. Thirdly, most metadata and datasets in Data.Gov are in English, thereby avoiding the multilingual issue, which falls outside the scope of this paper.

1) Table Repository: A snapshot of Data.Gov was taken on March 21st, 2023, capturing metadata for all datasets and tabular data resources. We first downloaded all the metadata with Data.Gov's CKAN API . Subsequently, we downloaded tabular data for each dataset with the urls in the metadata with a time budget of 60 seconds for each download. The Data.Gov snapshot comprised 247,675 datasets and 31,249 tabular data resources, of which 23,600 were successfully downloaded. Losses occurred due to (1) urls leading to landing pages instead of downloadable data files, (2) formatting issues with downloaded data files, and (3) errors during the downloading process. Our table repository includes all tabular data, along with their dataset-level metadata, from the Data.Gov snapshot.

2) Column Semantics Annotation: We then applied the prompt generator, PLM executor, and column semantics extractor to annotate column semantics for all the tables in our table repository. With GPT-3.5-Turbo [6] and dataset title and description as contextual information, we sent 8,407,413 tokens as prompts, and the model returned 6,071,182 tokens. The total cost was approximately 25 USD. Out of the 23,600 tables, we successfully parsed the model's response for 20,578 tables. Failed parsing occurred when (1) the model changed the column name in the response, (2) the model didn't follow the instruction and returned a badly-formatted response.

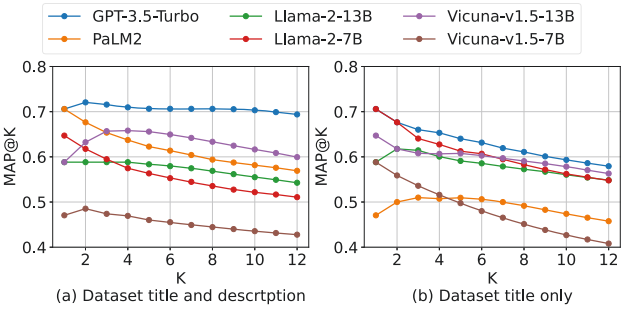

Fig. 3: Performance of ARTS on the benchmark with different PLMs using different set of metadata.

3) PLM Executor: A variety of language models are included in the demonstration plan. Specifically, our selection includes two closed source models, GPT-3.5-Turbo [6] and PaLM2 [7], and two open source model, Llama2 [8] and Vicuna-v1.5 [9]. For both Llama2 and Vicuna-v1.5, we use the 7B and 13B versions, which results in a total of 6 different models. For closed source models, we directly call their API for PLM execution. For open source models, we serve the models on our own machine with FastChat [10].

E. ARTS's Performance

To evaluate ARTS's performance, we curated a benchmark by manually annotating aggregate related tables from the table repository we developed using Data.Gov. For each of the query tables, we used the dataset title, dataset description, and dataset tags to find the top 100 tables in the repository by semantic similarity, thus a candidate table set with at most 300 tables were constructed, then we manually examined each to identify aggregate related tables.

Evaluation metrics. We employ metrics akin to other table search systems, utilizing the mean average precision at K ($MAP@K$) to assess the efficacy of the aggregate related table search problem [1], [3], [4].

Discussion. As depicted in Figure 3, GPT-3.5-Turbo performs best regardless of the metadata used. It surpasses other models by a remarkable 20% margin on $MAP@12$ when utilizing both the title and description. Vicuna-v1.5-13B claims the second position in terms of performance across both metadata configurations. In contrast, Vicuna-v1.5-7B has the lowest performance among all models. One hypothesis for this disparity is that the fine-tuning process of Vicuna is more effectively assimilated by the 13B model, especially given the relatively modest performance difference between the 7B and 13B version of Llama2. Intriguingly, models exhibit different behaviors with different metadata sets. Both PaLM2 and GPT-3.5-Turbo exhibit enhanced performance when incorporating both the title and description. On the other hand, models like Llama2-7B, Llama2-13B, and Vicuna-v1.5-7B maintain relatively consistent performances across metadata configura-

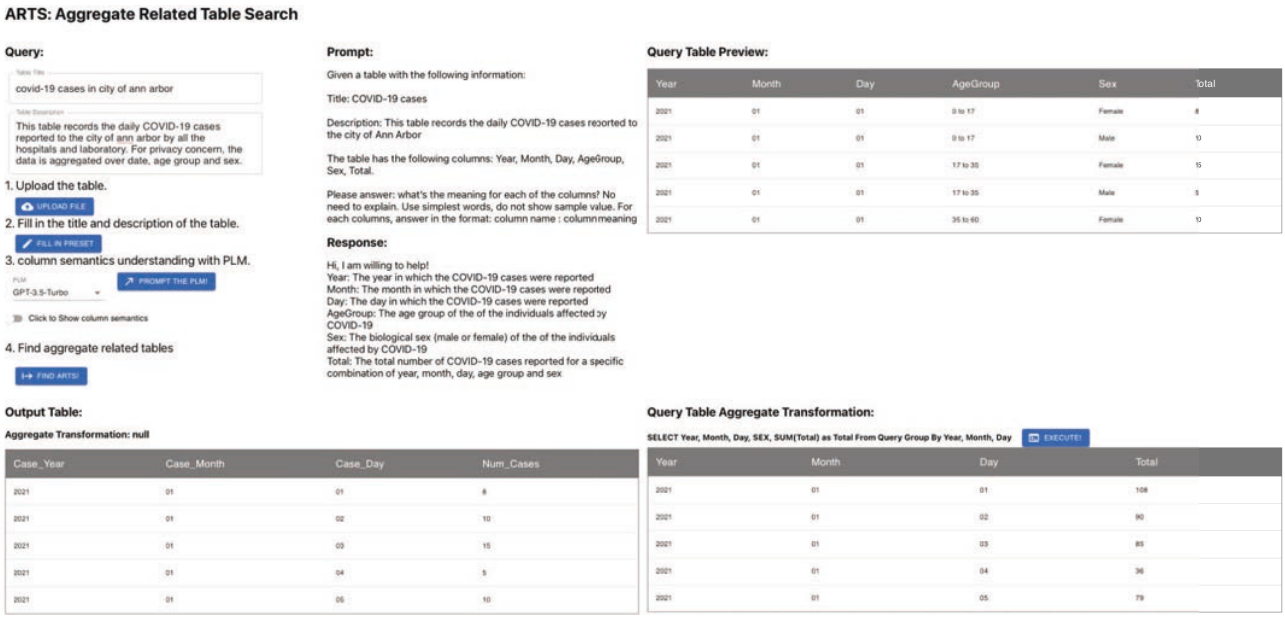

Fig. 4: ARTS's Web Interface.

tions. Some even perform slightly better than their counterparts when solely relying on the dataset title. These findings suggest that models have different proficiencies when confronted with longer context windows. While GPT-3.5-Turbo and PaLM2 benefit from expanded contextual information, the same cannot be asserted for other models. The result also demonstrates the effectiveness of ARTS in sparse context given that the average length of dataset title in the benchmark is only six words.

III. Experience

Our demonstration aims to show the conference attendees that our proposed system ARTS is able to retrieve aggregate related tables effectively, and the proposed aggregate related table promotes the usability of the table search system.

To best illustrate how to use our tool, we prepared three sets of query table, table title, and table description. Attendees can also manually enter their data.

To show how ARTS can help users find aggregate related tables, we use the motivating example from Fig. 1 where a user would do the following:

1) Click the "upload your table" button, and select the "covid_example_table.csv" as the desired file.
2) Set table title to: "covid-19 cases in city of ann arbor".
3) Set the description of the table to: "This table records the daily COVID-19 cases reported to the city of ann arbor by all the hospitals and laboratory. For privacy concern, the data is aggregated over date, age group and sex."
4) Click "Prompt the PLM". ARTS will generate the prompt, call the model's API, and extract column semantics from the response.
5) Click "Find ARTs!". ARTS will quickly process the request and return a list of tables as depicted in Fig. 4.
6) User can click on different tables to see the aggregate transformations in detail.

IV. Conclusion

Our demonstration showcases a novel search system aimed for aggregate related tables. Aggregate related tables requires a more comprehensive understanding of column semantics in a fine-grained fashion. With only limited contextual information from the user side, ARTS can find aggregate related tables effectively and efficiently. This demonstration is a good illustration of our vision on a more effective table search system that requires a minimum of user input and returns highly related tables.

Acknowledgment

This work was supported in part by NSF grants 1934565, 2106176 and 2312931.

References

[1] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," Proc. VLDB Endow., vol. 11, no. 7, pp. 813–825, 2018.
[2] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in SIGMOD Conference. ACM, 2012, pp. 817–828.
[3] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in SIGMOD Conference. ACM, 2020, pp. 1951–1966.
[4] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller, and M. Riedewald, "SANTOS: relationship-based semantic table union search," Proc. ACM Manag. Data, vol. 1, no. 1, pp. 9:1–9:25, 2023.
[5] https://data.gov, U.S. Government's Open Data.
[6] https://platform.openai.com/docs/models, GPT-3.5-Turbo.
[7] https://ai.google/discover/palm2/, PaLM2.
[8] https://ai.meta.com/llama/, Llama2.
[9] https://lmsys.org, Vicuna-v1.5.
[10] https://github.com/lm-sys/FastChat/tree/main, FastChat.