Table Discovery-Draft

1. Problem definition

Given a natural language question, the system needs to search among a large collection of tables and return the answers. (or the right table/row)

Given a natural language question q, and a collection of tables, T, we want to find a tuple (t, r, r.A) where t is a table, r is a tuple/row in T, and r.A is an attribute value in r. This corresponds to the answer to the question, the tuple where the answer is located, and the table where the tuple is located.

Given that tuple, we can measure precision and recall at 3 different levels. We can measure p@k and r@k at the table level, tuple level, and answer level.

In discovery scenarios, the hardest problem is to find the right table among T, e.g., find 1 table among thousands. Once the right table is in front of the analyst, it becomes a lot easier to find the desired answer there. Consequently, we design the system to maximize the retrieval quality of t \in T.

We have least assumptions on tables
1) There is no information (or text) on what a table is about.
3) There is no schema information about a table other than its column names and row cell values.
4) Column name should be interpretable by the user.
5) Multi-layer Columns

			Profit	
	2020 Income	2021	2020	2021
A	123,3333,			
B				

2. Current approaches

2.1 Full text search index

The most simple baseline is to index every tuple in every table in elasticsearch along with the row id and the table id, and then use the natural language question to find the relevant table.

2.2 Dense Table Encoder + Table Reading Comprehension (RC) Model

In OpenTableQA (Herzig et all., 2021) first a learned Table Encoder precomputes all tables into dense vectors. During query time, the question is also encoded into a question dense vector, and then a similarity score is computed between the question vector and a table vector, and top tables are thus retrieved. Then each (question, top table) pair is fed to the Table reader (Herzig et all., 2020) model to retrieve answers.

2.3 Query2SQL

This approach translates the question into a Database SQL statement and then issues the sql to a Database Management System (DBMS) to return answers

We likely won't be able to use this approach as it needs the full schema information during training. Let's make sure that's the case and let's discuss it in the related work.

3. Our approach

Our system first generates text from the tables using a generative model, Graph-to-Text (Ribeiro et all., 2020)   and then use a OpenQA system to return answers

3.1 Why this approach ?

1) Dense encoder approach Computation intensive
2) In Table Discovery , table2txt + Sparse Index is as good as or better than the Dense approach
3) This approach does not require training other than pre-trained components available off-the-self.

3.2 Contributions

1) New techniques in Data2txt (Table2txt)

3.3 Challenges

The NLP community has proposed a lot of models (references) to generate text from structured data including tabular data. However, directly applying those models in our case still faces many challenges. To clearly describe those challenges, we assume a table of the following form.

C_1	C_2	…	C_{m-1}	C_m
…		$R_i(C_j)$		
$R_n(C_1)$				$R_n(C_m)$

It has m columns, or attributes, C_1 … C_m and n rows, R_1 … R_n . We denote by $R_i(C_j)$ the cell value of row i and column j.

3.3.1 Graph Representation

To generate text from the table, Graph-to-Text (Ribeiro et all., 2020) requires an acyclic directed graph representation for each row where the vertices denoting the row cell values and edges denoting the attributes. For example, in a soccer ***Team*** table (Nan et all., 2021) with columns (***Team, Stadium, Capacity, Opened-Year, City***), the graph representation for any row R_i is shown in the following figure.

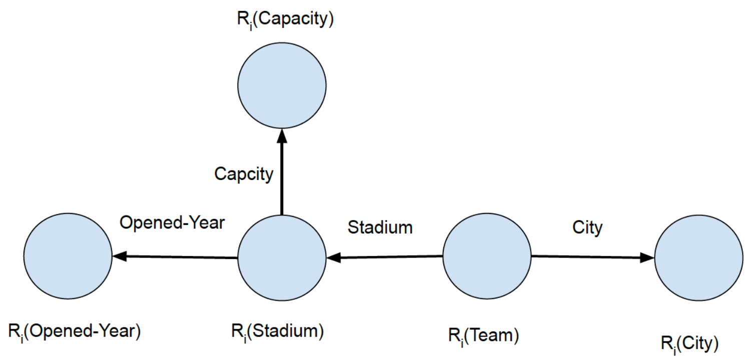

Figure 1, Graph representation for a table row

This graph actually shows the relations (dependencies) among columns. In our case (tabular data), the graph is actually a tree. However, we have no such graph as input.

One approach thus is to infer the column dependencies and then create the graph from the table which is called ***functional dependency discovery*** (Papenbrock et all., 2015, Zhang et all., 2020) in the database community. (evaluation is needed to discuss more about this approach)

Another approach is to include all possible column dependencies and thus construct a more complicated graph. Specifically, we add an edge from C_i to C_j for each j in [i+1, m]. This adds an edge between each two columns and makes sure there is no cycle. For simplicity , we ignore the difference between the edges (C_i, C_j) and (C_j, C_i) because we found not much difference in the generated text. This approach guarantees the correct dependencies are considered, but introduces more noise text because some dependencies are certainly not valid. An improvement over this approach is to consider all dependencies but then remove those that are more likely incorrect. Instead of identifying all functional dependencies, we could identify just ‘keys’. Then generate the dependencies considering those columns that are most likely keys. This should help prune out the dependencies, reduce the noise, and reduce the time to generate the text. We would need to compare these approaches though.

3.3.2 Noise text from incorrect column dependencies

This challenge is from section 3.3.1, currently not addressed yet.

3.3.3 Generation over large rows

Currently, we run Graph-to-Text (Ribeiro et all., 2020) row by row. In the case of large rows, this will take a long time. Although we can generate in parallel, a more promising solution is to generate a template from a small group of rows and then apply this template to the remaining rows. There are some simple approaches but they do not always work.
One simple approach is to generate the text from the first row, and then find a row cell value $R_1(C_j)$ in the text and replace it with $R_k(C_j)$ for the row k. However the generated text does not always include $R_1(C_j)$. This approach can be made more robust by generating text for n rows instead of only 1, and then trying to synthesize the template from the sample ‘n’.
Another naive approach is to use placeholders to generate text instead of the real $R_i(C_j)$ cell value, however the generated text is very bad and nonsense. It is clear the Graph-to-Text (Ribeiro et all., 2020) model depends on the cell value to generate good text and the column names together with the graph structure are not enough.

3.3.4 Generation over large columns

Although the number of columns won’t be as large as 1,000 in the real scenario. A table with 100 columns is not uncommon. In this case, the graph may contain more tokens than allowed for the model because of the limit token size of transformer-base models. So a chunk strategy is needed to limit the input graph. This partition strategy could rely on approximate ‘keys’, as explained above. We would need to make sure that each partition of attributes has at least one key, so the generated text makes sense.
If a question involves 3 attributes of the table, and these 3 attributes are in different chunks, we may not be able to answer the question. Unless we merge passages a posteriori (i.e., after receiving a question). For example, after receiving a question, no single passage may help answer it. But the IR system may still pull out the relevant passages. Merging them on-the-fly may help in some cases. In any case, this seems like an optimization we should only consider based on data.

OpenTableQA (Herzig et all., 2021) should have the same issue as well.

3.3.5 Consistency of generated relation across rows

Given a column, for any two rows, the generated text should imply the same relationship, should we double check the consistency ?
If we want to generate a template from a small group of rows as in section 3.3.3, we may need to check this, i.e. we need to come up with a reasonable text from this small group first. Great idea. I explained above we would likely need to generate templates from 'n' rows instead of only 1. What you are suggesting here is how to use the sample, and it sounds like we should give this a try.

4. Evaluation

4.1 Dataset

For each dataset, we need both Table Corpus and Questions.

Right now we only have one, i.e. ***NQ-TABLES***. This dataset is from ***OpenTableQA*** (Herzig et all., 2021). It contains 210,454 tables and 990 test questions. There are, in addition, 13,740 train questions and 1,524 dev questions, but we cannot use them if we want to compare our system to ***OpenTableQA*** (Herzig et all., 2021) which is trained and fine-tuned on those train/dev questions.

Another work that is very close to ours is ***TableCellSearchQA*** (Sun et all., 2016) from Microsoft, but the dataset is not available.

WTR (Chen et all., 2021) releases its dataset which is built on "the English subset of WDC Table Corpus 2015 which includes 50.8M relational HTML tables extracted from the July 2015 Common Crawl". The table corpus has more topics than the one in ***NQ-TABLES***. However the questions (I would say queries) are not asked to retrieve table cells but the whole table. e.g. "usa population by state", "ipod models", "football clubs city". Also it only contain 60 queries.

OTT-QA (Chen et all., 2021) contains table corpus and questions, but most of the questions depend on surrounding text as well, while we require questions to be answerable over tables. There may be some questions (one-hop) satisfying our scenario, but needs to be identified heuristically or by hand.

Another possible dataset is from Orihuela et all., 2021

In summary, it is easy to find a table corpus but more difficult to find answerable questions over the table corpus.

4.2 Metrics

It depends on the question types

1) The answer to the question is one or more whole tables

1.1) Metrics

e.g. "what is the usa population by state?". In this case we need to measure the retrieval performance. The metrics are

a) P@K

Precision in top K tables

b) NDCG@K

Normalized Discounted cumulative gain in top K tables

c) MAP

Mean Average Precision

These 3 metrics are used in ***WTR*** (Chen et all., 2021).

1.2) Baselines

a) ***OpenTableQA*** (Herzig et all., 2021)

b) Baselines in ***WTR*** (Chen et all., 2021)

The paper evaluates (summaries) state-of-the-art retrieval models, but they rely on surrounding text, and it takes more time to redo in our case

c) Different strategies of our approach

2) The answer to the question is one or more table cells

In addition to P@K, NDCG@K and MAP, we also need to evaluate EM and F1 w.r.t. the gold truth answers.

2.1) Metrics

a) EM

b) F1

2.2) Baselines

a) ***OpenTableQA*** (Herzig et all., 2021)

b) ***TableCellSearchQA*** (Sun et all., 2016)

Hope we can implement their model

4.3 RQ1: Table Retrieval Performance

4.3.1 Setup

1) Dataset: NQ-Tables

2) Baseline

2.1) OpenTableQA (Herzig et all., 2021)

2.2) TableTokenText + bm25

For each table, concatenate column tokens, and row cell tokens to get a text document, and then index all those documents.

4.3.2 Result

1) All tables (169,898)

System	Dev (rerun)		Test (rerun)		Test(reported)	
	P@1	**P@5**	**P@1**	**P@5**	**P@1**	**P@5**
OpenTableQA (Herzig et at., 2020)	40.96	68.60	44.21	73.62	42.42	-
TableTokenText + bm25	15.18	25.77	13.87	22.73		

2) Part of tables (21,642)

System	Dev		Test	
	P@1	**P@5**	**P@1**	**P@5**
OpenTableQA (Herzig et at., 2020)	**60.26**	**87.25**	**68.09**	**90.82**
TableTokenText + bm25	25.40	40.02	23.04	36.39
Graph_to_text + FabricQA	32.71	47.99	31.18	44.73
Graph_to_text (including table name) + bm25	35.15	54.83	37.64	54.64
Graph_to_text (including table name) + FabricQA	**46.67**	**71.88**	**45.36**	**69.76**

OpenTableQA + Other Dataset ?
Tricks to reduce gap

1) Data driven (borrow tricks from other papers or own tricks)

Retrieve same tablerss as

Dirty tables ?

1) Crawl parser
2) Error schema

table2txt over dirty tables (contribution)

4.4 RQ2: Cell Retrieval Performance

System	Passages	EM	F1	Oracle EM	Oracle F1
OpenTableQA (Herzig et at., 2020)		37.69	47.7	48.2	61.5
Graph-to-Text + FabricQA	30	14.75	20.59	32.02	44.21
	50	14.85	20.72	36.57	49.79
	150	14.95	20.87	41.21	56.53

5. Question Analysis in Table Retrieval

We analyze 100 questions where OpenTableQA is correct in finding the gold tables at top-1 while Grapg2txt&FabricQA is not. We group the possible reasons into 9 categories and the breakdown is shown in Figure 2 below.

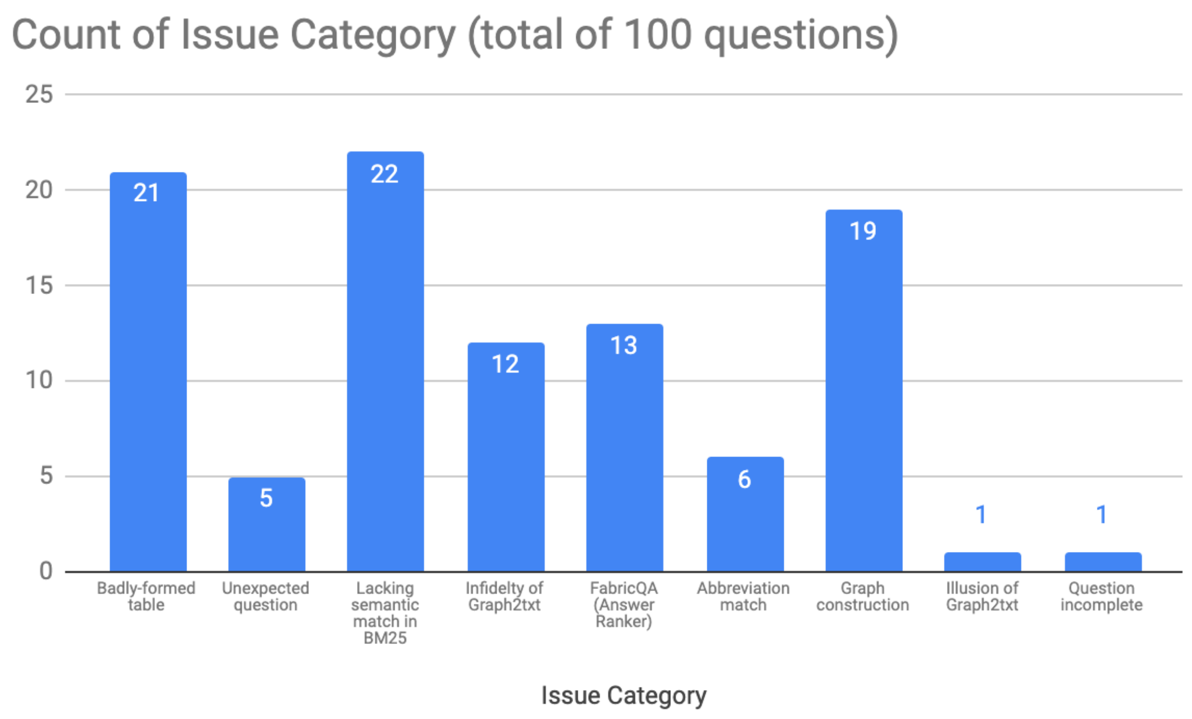

Figure 2, Issue categories

We now discuss each category in detail

5.1 Badly-formed table (21%)

Description
1) The answer column is empty, or just a copy of one row cell of that column. This causes meaningless relationships.
2) The row cell of the answer column contains a large block of text which needs further processing. Currently, If it is too long it will be truncated to a fixed maximum size.

Solution
We need to assume some well-formedness of the tables. So
1) We can fix those gold columns but it takes time and also this leads to bias (the non-gold table is still badly-formed)
2) Ignore those questions with badly-formed gold tables and evaluate on the remaining
3) Use another cleaner dataset.

E.g. The datasets in the data portal of Chicago, New York, etc are very clean (We can also add noise to it if needed) and representative (hierarchical columns), we can try to download them automatically and ask (by Amazon Mechanical Turk) questions over them. Possibly others already do these.

Of course we can continue to try datasets in other papers.

5.2 Unexpected question (5%)

Description
1) Questions are asked about the aggregation information of some columns.
E,g,
"how many songs are on vices and virtues"
"which one is the biggest city in usa"
Graph2txt generates text from cells and columns. So these aggregation keywords can not be matched unless some aggregation result is explicitly in some cells.
2) List questions
E.g.
"agatha christie witness for the prosecution bbc cast"
The answer should return all players in the show/movie
FabricQA are trained on cell questions which usually contain "what", 'when' and 'who' etc, So i may not be good at locating the right passage(table)

Solution
For 1), The model should know that to answer the aggregation question, the aggregation keywords do not necessarily exist in the passage (table).
For 2), we may borrow ideas from **Semantic Table Retrieval Using Keyword and Table Queries(**Zhang et all., 2021**)** which actually assumes such questions.

5.3 Lacking semantic match in BM25 (22%)

Description
For these questions, the top 100 passages returned by Elasticsearch do not contain the gold table. The gold table contains different words than the question words but semantically close and can answer the question.

Solution
Simply retrieving more passages or making sure there are at least K tables (calling multiple times) returned from Elasticsearch will greatly impact the throughput.
So we need to use Dense Index or combine Sparse Index with Dense Index, both are studied a lot by others so we can just use a off-the-shelf one.

5.4 Infidelity of Graph2txt (12%)

This is also a research problem in the NLP community.
Description
1) Graph2txt takes a row graph as input which contains a lot of (S, R, O) triples, however it ignores the gold triple for some graph and thus the generated text cannot answer the question. One explanation is that it may trade fidelity for fluency.
2) Graph2txt does translate the gold triple but does not preserve the original meaning, and in some cases, the result may be far away.
E.g. the column name “enter office” is decoded into “was born”

Solution
A possible approach is “Controlled Generation”. E,g, the prompt technique. Further papers may be read on this topic to discuss more.

5.5 FabricQA (Answer Ranker) (13%)

Description
The generated text (table) is good and should be ranked at top 1 but not by FabricQA.

Solution
We may try other state-of-the-art passage(answer) rankers(retrievers).

5.6 Abbreviation match (6%)

Description
In this scenario, the table name is critical to match the question, however the question or table name (not both) contains abbreviations and it is thus difficult to match.

Solution
1) Dense index probably does not work here. **Table Cell Search for Question Answering(**Sun et all., 2016**)** relies on 3rd database to link the abbreviation (alias) to the full entity name.
2) A naive approach, the abbreviation is often acronym, we may heuristicly create acronym for entities

5.7 Graph construction (19%)

Right now , this is about how to integrate the table name, column name in the graph which is the input of Graph2txt.
Description
1) Table Name integration
Currently, text is first generated for each row graph, and then words in the table name are inserted (separated by period) before the generated text to get the passage. This simple

approach is just to avoid rerunning the slow text generation from tables. Although it does help compared to ignoring table names, there may be better ways.
When we inspected some of the gold tables, we found the table name often contains the key entity and the table content contains the attributes along some other dimension (e.g. time) or its components. Without including the key entity the generated text from table columns and cells are not that meaningful.
E.g. given the question "**what is the area code for cincinnati ohio**" and the gold table with table name "**List of Ohio area codes_5F41C5A46B3096B0**" as below

Code	Created	Region
216	1947	Cleveland (October 1947)
234	2000	Akron, Canton, Youngstown, and Warren, overlay with 330
330	1996	Akron, Canton, Youngstown, and Warren, overlay with 234
...	...	...

Right now the generated text for the 1st row is,
"**216 was created in 1947 in cleveland (october 1947) .**"
After the table name words are inserted, the passage is
"**List of Ohio area codes 5F41C5A46B3096B0. 216 was created in 1947 in cleveland (october 1947) .**"

A more logic text would be
"**The Ohio area code 216 was created in 1947 for Cleveland(october 1947)**"

2) Subject Column Name integration
Currently, we ignore the column name for the subject, i.e. we feed ("216", "Created", "1947") to Graph2txt, It would be better to include the subject column name as ("code 216", "Created", "1947").

Solution
1) For table name, We can add another pseudo column to the table with cell value for all rows to be the key entity in the table name and then connect all columns to get a row graph.

Inspired idea, In addition, the table name may help simplify the graph. Instead of connecting all columns, we can try to identify which candidate key column is most possibly the real key column (assuming only one). By candidate key columns we mean the column which has the most distinct values. E.g. A table has 6 Columns C1, C2, C3, C4, C5, C6 and N rows. C1, C2, C3 all have N distinct values, while C4, C5, C6 have less than N values. So C1, C2, C3 are candidate key columns. We then use word2vec (this may be domain specific and thus not good in other domains) to measure which of the candidate columns is closest to the entity in the table name. Then the closest one (satisfying some threshold) is the key column. We then only use this key column to connect all other columns and see the result. If no close candidate column is found by the threshold, we then add a pseudo column to the existing table columns and make the entity in the table name as the cell values of that pseudo column. This approach may not work in hierarchical columns as in Figure 1, but should be a good baseline.

2) For the subject column name, just concatenate with the cell values.
We have not tried and hope this will be better.

5.8 Illusion of Graph2txt (1%)

This is also a research problem in the NLP community.
Description
This is similar to ***Infidelity of graph2txt.*** Graph2txt generates text (relationship) not implied in a table (let's call it table A). However it happens to be the right passage(answer) for the question. Although the gold table (let's call it table B) also generates good passage, FabricQA then chooses table A instead of table B This happens in low probability but it is a problem

Our choice to use the complete graph for columns can also have the illusion problem.

Solution
Not much idea of what the techniques in NLP are. More papers on this topic need to be read, but less of priority.

5.9 Question incomplete (%1)

Description
Some important words are missing in the question, e.g.
“when does life is strange before the storm part 2”
It actually asks when the part 2 of the movie “life is strange before the storm” was released and the gold table contains the column “release”. FabricQA should be able to find the gold table because the top-1 passage is not that meaningful.

Solution
Incomplete questions are usual when we use google. So we may try to make our system more robust.
Another experiment with the correct question (including “release”) should be conducted to verify the reason.

6. Reference

1) Zhang et all., 2021 , **Semantic Table Retrieval Using Keyword and Table Queries**
Section 2.1 (Table search) is a very good summary.
It assumes Web Tables where more context information is available, e,g, Table

page title, Table caption , etc, but not sure how much this extra information contributes to the result.
2) Sun et all., 2016 **Table Cell Search for Question Answering**
3) Shraga et all., 2020 , **Web Table Retrieval using Multimodal Deep Learning**
Informative table properties
a) Description (page title, table caption)
b) Schema (column names)
c) Row data

a and b are useful for list questions like “what are the area and population of continents?”

4) Zhang et all., 2020 web , **Web Table Extraction, Retrieval, and Augmentation: A Survey**
5) Ho et all., 2021, **QuTE: Answering Quantity Queries from Web Tables**
6) Chen et all., 2021 , **WTR- A Test Collection for Web Table Retrieval**
7) Hulsebos1 et all., 2021 , **GitTables: A Large-Scale Corpus of Relational Tables**
8) Pimplikar et all., 2012 **Answering Table Queries on the Web using Column Keywords**
9) Herzig et all., 2021 , Open Domain Question Answering over Tables via Dense Retrieval
10) Herzig et all., 2020 , TAPAS: Weakly Supervised Table Parsing via Pre-training
11) Yin et al., 2016 , Neural enquirer: Learning to query tables in natural language
12) Ribeiro et all., 2020 , Investigating Pretrained Language Models for Graph-to-Text Generation
13) Nan et all., 2021 , DART: Open-Domain Structured Data Record to Text Generation
14) Zhang et all., 2020 , A Statistical Perspective on Discovering Functional Dependencies in Noisy Data
15) Papenbrock et all., 2015 , Functional dependency discovery: an experimental evaluation of seven algorithms
16) Orihuela et all., 2021, Natural Language Inference over Tables: Enabling Explainable Data Exploration on Data Lakes
17) Chen et all., 2021 Open Question Answering over Tables and Text

A. Ideas and questions

A.1 Table BM25

Instead of counting the frequency of tokens in documents, what if we compute the frequency of tokens w.r.t tables. Sounds like a great idea. We would need to compare both approaches to be able to test the hypothesis.

A.2 Table declares what questions can be answered?

Since we have the table structure, each table can generate some question templates that can be answered. We can match the question to the question templates to boost the confidence of the matched tables.

This needs training or tuning, but it is a new idea. Yes, although I would hold off from introducing any component that needs training now.