A Hybrid Approach to Functional Dependency Discovery

CS 520: Literature Review

Hrishika Shelar | Supriya Hinge | Jay Patel

Introduction

1. What is functional dependency?
   - pairs of records with the same values
   - Represented by $X \rightarrow A$
   - implies that the values in A are functionally dependent on the values in X
   - For example, a person's birth date determines the age, zip code determines the city, etc.

2. Use of functional dependencies
   - schema normalization
   - data integration
   - data cleansing and other data management tasks.

3. Problems with identifying functional dependencies
   - functional dependencies are usually unknown and almost impossible to discover manually
   - Most algorithms cannot process datasets of real world size

Problem

- Database research has offered many discovery algorithms to find functional dependencies of a dataset which are unable to work on real world size datasets which contain more than 50 attributes and millions of records.
- Discovery Algorithms are needed to reveal all the functional dependencies in a dataset and should be able to optimize both records and attributes.
- Finding functional dependencies is quadratic to the number of records (rows) and exponential to the number of attributes (columns)

Related Work

- Paper refers to strengths and weaknesses of several different algorithms for solving functional dependencies problem
- Some algorithms have inclined towards approximation and conditional functional dependencies to over come complexity
- Some algorithms have adapted parallel & distributed processing for discovery
- Lattice Traversal Algorithms performs well with large records but fails to scale as number of columns increases due to candidate-driver search strategy.
- HyFD derives FD pruning rules from Lattice Traversal Algorithm
- Difference- and agree-set algorithm takes subset of columns, and created FD.
- It creates agree set with valid FDs and difference set with invalid FDs. Goal is to maximize agree set and minimize difference set.

Related Work

- Literature compares several algorithms for retrieving functional dependencies.
- Some Algorithms rely on Approximation & Conditional FDs
- Certain algorithms scale well row-wise, while others scale well column-wise
- Several algorithms use parallel & distributed computing to achieve faster results.

Lattice Traversal

- Performs well with **Large Datasets**
- Fails to scales to number of columns
- HyFD derives FD Pruning from this Algorithm

Difference - & Agree - Set

- **DEP-Miner** & **FastFD** algorithm analyze data for set of attributes pairs that agree on values.
- Goal is to **Maximize agree set** and **Minimize Difference Set**.

Dependency Induction

- FDEP analyze all records pairwise to find invalid FDs
- Discovery strategy is proven to scale well with high number of attributes

Sampling Based Hybrid FD Discovery

- We can calculate the functional dependencies of a small subset of a Dataset R such that, all values of the subset r' belong in the dataset R.
- Since the size of r' is smaller than R, the calculate of FD on r' is much cheaper than on R.
- FDs of r' can be valid or invalid
- they will exhibit 3 properties.
    - **completeness**: if all FDs of r' implies all FDs of R. [1]
    - **minimality**: if there is a minimal FD in r' than that FD is minimal in R as well.
    - **proximity**: if a minimal FD is invalid in r' there is a chance that it's close to specialization that are valid in R.

Why Hybrid Approach?

- All the algorithms in the related work either address scaling with increasing number of rows, or scaling with increasing number of columns.
- No previous work promise to provide the “best of both world”
- Therefore the hybrid approach provides column-efficient FD induction techniques with row-efficient FD search techniques.
- Processes in divided into two recursive parts:
  - Phase 1:
    - We take a subset of data from the input, and apply column efficient induction techniques. The goal of this phase is to produce low effort set of FD cancidates that are according to property 3.
    - The paper proposed focused sampling techniques that allows algorithm to select samples with possibly large impact on the result precision.
    - As we expect either property 1. or 2. to hold true in each sampling, we will never have incomplete result.
  - Phase 2
    - The algorithm uses row-efficient FD search techniques to validate the FD candidates that are passed by phase 1.

Solution

1. **Approach**

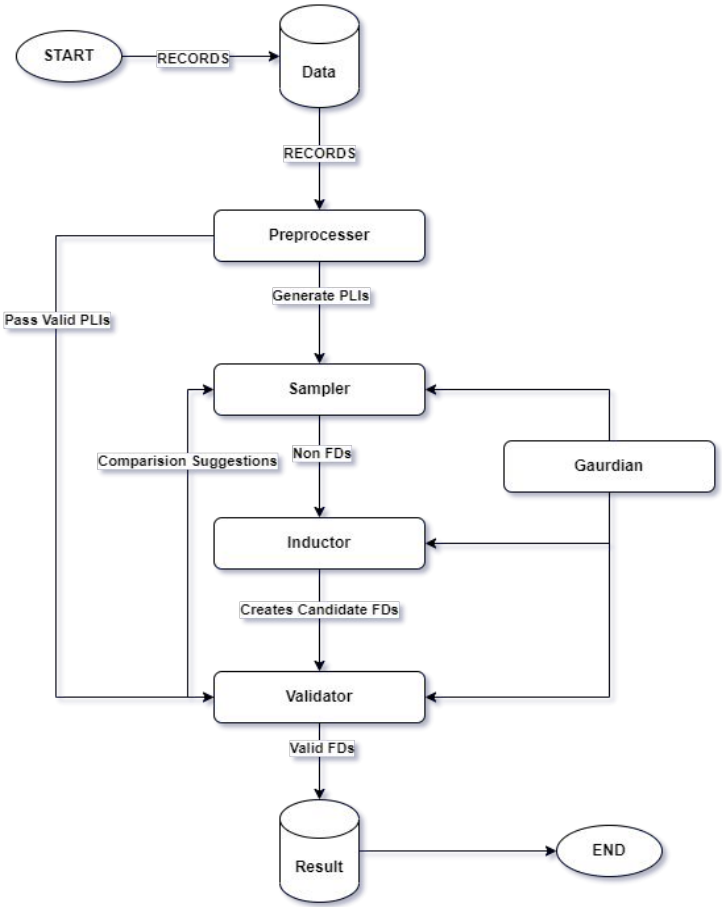
- A hybrid discovery algorithm called HYFD, which is a combination of approximation techniques and efficient validation techniques to find all the minimal functional dependencies in a given dataset.
- HYFD outperforms all the existing approaches while operating on compact data, and also scales to much larger datasets.

Solution

2. Algorithms

- HYFD algorithm is a combination of row and column efficient discovery techniques.
- It works in two phases, in the first phase, HYFD extracts a small subset of records from the input dataset and finds functional dependencies in this sample. The result of this phase is a set of FDs that are either valid or almost valid with respect to the complete dataset.
- In the second phase, HYFD validates the discovered FDs on the entire dataset and removes FDs that are not efficiently holding on to the entire dataset.
- If the validation is inefficient, i.e The resulting FDs do not apply to the entire dataset, HYFD is able to switch back to the first phase and continue the process again keeping in mind the previous FDs which were not applicable to the entire dataset.
- To conclude, HYFD is basically an alternating two phase discovery strategy which discovers minimal FDs in a given dataset.
- The HYFD algorithm is divided into five main parts, namely -
    - Preprocessor
    - Sampler
    - Inductor
    - Validator
    - Guardian

Flowchart


START
RECORDS
Data
RECORDS
Preprocesser
Generate PLIs
Pass Valid PLIs
Sampler
Comparision Suggestions
Non FDs
Gaurdian
Inductor
Creates Candidate FDs
Validator
Valid FDs
Result
END

Algorithm

1. **Preprocessor**
   - Transforms records into position list indexes (PLI).
   - Using PLIs, the records of the datasets are compressed
2. **Sampler**
   - Implements column efficient FD induction technique
   - Checks compressed records for FD violations.
   - Example - In schema R(A,B,C) records r1(1,2,3) amd r2(1,4,5) exist. A values match, B and C differ which makes r1 and r2 non- FDs
   - Cluster sampling technique is used.
3. **Inductor**
   - Converts non-FDs to FDs passed by Sampler Process.
4. **Validator**
   - Implements row efficient technique.
   - FDs from Inductor are taken as input and validated against the entire dataset.
   - If FDs are invalid, the process switches back to Sampler.
5. **Guardian**
   - Prunes the FD tree if necessary i.e., when FD tree grows and more memory is consumed.

Experimental setup

- **Metanome**, the data profiling framework is used for common task like input parsing, result formatting, and performance measurement are standardized by the framework and decoupled from the algorithms
- for null semantics, experiment use null = null, because this is how related work treats null values

- **Experiment 1** : Varying the number of rows
  - Measures the runtime on different row numbers using the ncvoter and uniprot dataset
  - The reason why HyFD performs so much better than current lattice traversal algorithms is the fact that the number of FD-candidates that need to be validated against the many rows is greatly reduced by the Sampler component.

- **Experiment 2** :Varying the number of Columns
  - Measures the runtime on different Column numbers using the Plista and uniprot dataset
  - In experiment 2- HyFD outperforms all existing algorithms.
- Two datasets used are 1,000 rows so small that comparing all pairs of records is feasible and probably the best way to proceed but HYFD does not compare all record pairs.

Evaluation

- HYFD is quadratic in the number of records n and exponential in number of attributes m same as FD discovery.
- Potentially each phase can discover all minimal FDs without the other.
- Experimental setup has shown that HyFD is able to process significantly larger datasets than state-of-the-art FD **discovery** algorithms in less runtime.

Dataset	Cols [#]	Rows [#]	Size [KB]	FDs [#]	TANE [12]	FUN [18]	FD_MINE [25]	DFD [1]	DEP-MINER [16]	FASTFDS [24]	FDEP [9]	HYFD
iris	5	150	5	4	1.1	**0.1**	0.2	0.2	0.2	0.2	**0.1**	**0.1**
balance-scale	5	625	7	1	1.2	**0.1**	0.2	0.3	0.3	0.3	0.2	**0.1**
chess	7	28,056	519	1	2.9	1.1	3.8	1.0	174.6	164.2	125.5	**0.2**
abalone	9	4,177	187	137	2.1	0.6	1.8	1.1	3.0	2.9	3.8	**0.2**
nursery	9	12,960	1,024	1	4.1	1.8	7.1	0.9	121.2	118.9	46.8	**0.5**
breast-cancer	11	699	20	46	2.3	0.6	2.2	0.8	1.1	1.1	0.5	**0.2**
bridges	13	108	6	142	2.2	0.6	4.2	0.9	0.5	0.6	0.2	**0.1**
echocardiogram	13	132	6	527	1.6	0.4	69.9	1.2	0.5	0.5	0.2	**0.1**
adult	14	48,842	3,528	78	67.4	111.6	531.5	5.9	6039.2	6033.8	860.2	**1.1**
letter	17	20,000	695	61	260.0	529.0	7204.8	6.0	1090.0	1015.5	291.3	**3.4**
ncvoter	19	1,000	151	758	4.3	4.0	ML	5.1	11.4	1.9	1.1	**0.4**
hepatitis	20	155	8	8,250	12.2	175.9	ML	326.7	5576.5	9.5	0.8	**0.6**
horse	27	368	25	128,727	457.0	TL	ML	TL	TL	385.8	7.2	**7.1**
fd-reduced-30	30	250,000	69,581	89,571	**41.1**	77.7	ML	TL	377.2	382.4	TL	513.0
plista	63	1,000	568	178,152	ML	ML	ML	TL	TL	TL	26.9	**21.8**
flight	109	1,000	575	982,631	ML	ML	ML	TL	TL	TL	216.5	**53.4**
uniprot	223	1,000	2,439	>2,437,556	ML	ML	ML	TL	TL	TL	ML	**>5254.7**

Results larger than 1,000 FDs are only counted **TL**: time limit of 4 hours exceeded **ML**: memory limit of 100 GB exceeded

Table 1: Runtimes in seconds for several real-world datasets (extended from [20])

Conclusion & Future Work

- In this paper, proposed HYFD algorithm discovers all minimal, non trivial FDs in relational database because it combines both row and column efficient discovery techniques.
- HyFD is the first algorithm that can process relevant real-world size datasets which contains more than 50 attributes and a million records.
- HYFD offers the fastest runtimes and the smallest memory footprints for small datasets.
- A task for future work is to develop use-case specific algorithms that leverage FD result sets for schema normalization, query optimization, data integration, data cleansing, and many other tasks.