Proceedings of the 21st International Conference on Global Research and Education (Inter-Academia 2024)

Functional Dependency Extraction with Sequential Indexed Search Trees

B. Tusor[a,b,*], A.R. Várkonyi-Kóczy[a,b] and S. Gubo[b]

[a]*Software Engineering Institute, John von Neumann Faculty of Informatics, Óbuda University, Bécsiút 96/b, H-1024 Budapest, Hungary*
[b]*Department of Informatics, J. Selye University, 3322 Bratislavská cesta, 945 01 Komárno, Slovakia*

Doi: 10.12693/APhysPolA.146.515 *e-mail: tusor.balazs@nik.uni-obuda.hu

Functional dependency analysis is an important field of data science, where the goal is to determine the relationships between different data attributes and attribute sets in a given data set. This can lead to gaining valuable information about the data that is often not evident through surface-level analysis. In previous work, the authors proposed a functional dependency extraction method called Sequential Indexing Tables, which is a specialized variant of the Sequential Fuzzy Indexing Tables (SFITs) classifiers. SFITs combined lookup table classifiers with fuzzy logic to implement a very fast yet flexible classification. A special feature of the SFIT classifier is that its structure indicates the functional dependencies between the data attributes that are present in the training data set. However, the main disadvantage of SFITs is that they require a significant part of the problem space to be stored in the computer memory, scaling exponentially with the number of attributes. To solve this issue, a new classifier called Sequential Fuzzy Indexed Search Trees (SFISTs) has been proposed by the authors, which builds on the same idea as SFITs but uses a more compact structure while providing a slightly better classification accuracy. In this paper, the functional dependency detection and extraction method is presented, which is a specialized version of the SFISTs classifier that uses the same base idea as its predecessor but with a much smaller spatial complexity.

topics: functional dependency (FD) extraction, data analysis, data science, indexing tables

1. Introduction

Functional dependency (FD) analysis is an important field of data science where the goal is to determine the relationships between different data attributes and attribute sets in a given dataset. This can lead to gaining valuable information about the data that is often not evident through surface-level analysis, most often used in database management (schema normalization [1], query optimization [2], data cleansing [3], etc.), but also can be used to evaluate the classification feasibility of classification models [4], data mining [5], etc.

In simple terms, attribute A (the dependent) is functionally dependent on a set of attributes X (the determinant) over a dataset ($X \to A$) if the dataset does not contain any two tuples that have the same value in the attributes of the determinant set attributes, but different values in the dependent attribute. The discovery process of FD generally consists of two steps: candidate generation and FD validation, which has a complexity of $\mathcal{O}(P^2N^2 2^N)$ [6], where P is the number of data tuples (the rows of the data), and N is the size of the schema (the number of attributes, i.e., the columns). Most of the methods proposed in the literature attempt to mitigate the quadratic dependence on the number of tuples by minimizing the necessary number of data comparisons with various heuristics.

In previous work, the authors proposed a new classification method called *Sequential Fuzzy Indexing Tables* (SFITs) [7], combining lookup table classifiers with fuzzy logic to make a very fast yet flexible classifier. A special feature of the SFIT classifier is that its trained structure implicitly indicates the presence of functional dependencies between the data attributes in the training data set, meaning that there is no need for data sample comparisons, thus reducing the computational complexity of the FD discovery problem. Thus, the FD discovery method *Sequential Indexing Tables* (SITs) [8, 9] was developed, which has a very high operational speed, but, in turn, similarly to the SFIT classifier, it requires a significant part of the problem

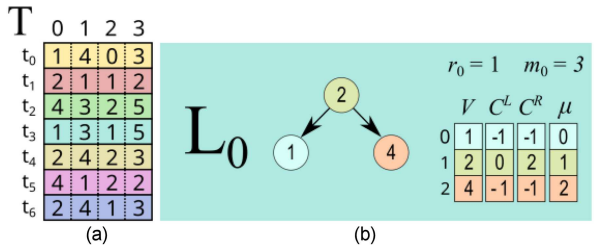

Fig. 1. An illustrative example of an integer-valued dataset with 4 attributes (a), and the layer built using attribute #0 (b). For the latter, the search tree and its indexed array representation are shown.

space to be stored in the computer memory, making it less usable for higher-dimensional problems as its structure size scales with the size of the known value domain of the attributes. To solve this issue, a new classifier called *Sequential Fuzzy Indexed Search Trees* (SFISTs) [10] has been proposed by the authors, which builds on the same idea as the SFITs but uses a more compact structure (using balanced search trees), while providing a slightly better classification accuracy.

In this paper, a new functional dependency extraction method called *Sequential Indexed Search Trees* (SIST) is presented, which is based on the SFISTs classifier. The main goal of this preliminary research is to explore the capabilities and applicability of the proposed method, alongside its weak points, to figure out which aspects of the method need further development.

The rest of the paper is as follows. In Sect. 2.1, the FD indication of the proposed method is described in detail, while in Sect. 2.2, the evaluation sequence generation (in which the attribute combinations are processed) is presented. In Sect. 2.3, the steps of the FD extraction are summarized, in Sect. 2.4, the experimental results are shown, then Sect. 2.5 examines the computational and spatial complexities of the SIST method, and Sect. 2.6 summarizes the gained experiences. Finally, Sect. 3 concludes the paper and presents future work.

2. Sequential Indexed Search Trees

2.1. Functional dependency indication

Similarly to its predecessor (the aforementioned SITs method), instead of comparing tuples to each other to find FDs, the SIST extractor analyzes value combinations by building a sequence of indexing arrays, sacrificing memory usage for faster operation.

In each layer in the layered architecture, the sets of tuples of the dataset are virtually separated into subsets, based on their corresponding attribute value. Each subset has its own subset index, and in the next layer, the tuples in each subset are further divided (again, by their corresponding attribute value). If the number of subsets in layer i is the same as in the previous layer, then the value cannot divide the subsets any further, which means that for all value combinations in the previous layer, there is only one value in attribute A_i, so $X \to A$ holds (where X is the set of the attributes of the layers that precede A_i). This presents a very simple and fast way to check if an attribute is functionally dependent on one or more other attributes, namely by simply building the layered structure using the supposed determinant set and building a layer using the dependent set upon them.

In each layer i, the data is stored in the form of a semi-balanced binary search tree, where each node corresponds to a distinct value in A_i (from the given dataset). Let S_V denote the number of nodes. The tree is implemented as a series of S_V-long arrays, where

- V stores the given value,
- C^L stores the index of the left child of a node,
- C^R stores the index of the right child of a node.

The index of the root node is stored in r. Given that it is a binary search tree, any given value can be found in the tree simply by starting at the root node and then proceeding to the left or right child, depending on the given value being smaller or bigger than the value of the node, and stopping if the values are the same.

Furthermore, the aforementioned subset indices are stored for each node in the 2D array μ, for which the number of rows is S_V, while the column number depends on the number of subset indices from the next layer. For the first layer, the latter is 1. The number of indices in a given layer is stored in m.

While V, C^L, and C^R are constants (they are built only once for each attribute), the index array is created dynamically for each FD investigation.

Normally, in order to gain this subset index from the previous layers, each layer would need to be evaluated again, so to avoid this computation overhead, $P \times N$ sized 2D array H is maintained, where $H_{k,i}$ stores the subset index value tuple k gained in layer i.

Figure 1a shows an example of a simple dataset T with 4 attributes and 7 tuples, while Fig. 1b illustrates the first layer built from the data, i.e., the binary search tree and the arrays that implement the tree. Since the data of attribute $i = 0$ consists of values 1, 2, and 4, they will be the contents of the value array. The root of the tree is at index 1, so indices 0 and 2 will be the left and right children of the tree, respectively. Each tuple t_k is processed in the dataset, but firstly, the index of the value array that equals their own value ($t_{k,i}$) is acquired (in $\log_2(S_V)$ steps). If the corresponding μ value is -1, then the value is updated to the current number of

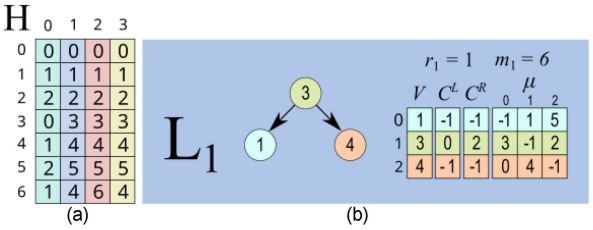

Fig. 2. Following the previous example, (a) H is built to store the corresponding μ value for each tuple in each layer, and (b) the second layer (for attr.#1) is built upon L_0.

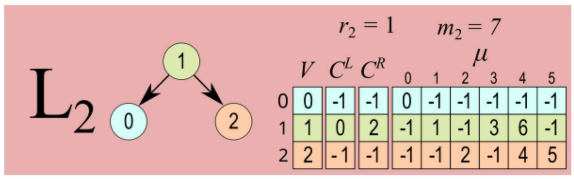

Fig. 3. A layer built on L_1 using A_2.

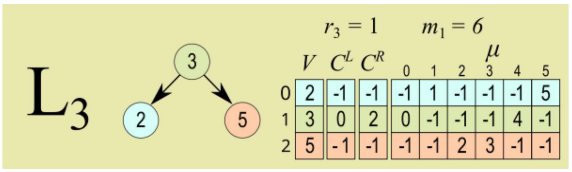

Fig. 4. A layer built on L_1 using A_3.

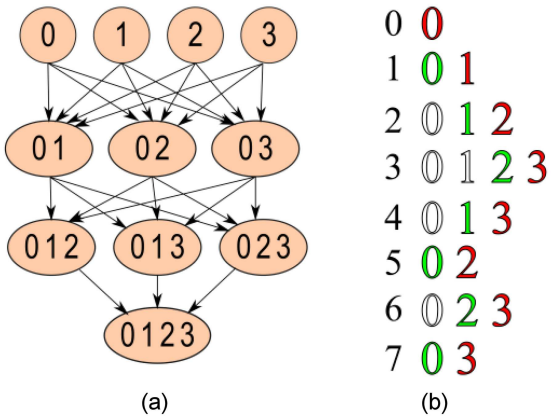

Fig. 5. (a) A set containment lattice for 4 attributes, and (b) an evaluation sequence for the lattice nodes (that start with 0).

indices (m_0), and m_0 is incremented. If the value is larger than -1, then the gained subset index value is noted in H (Fig. 2a).

In the next layer (Fig. 2b), each tuple k is sorted into a subset based on its value (indicating the row) and its subset value acquired in layer 0, stored in $H_{k,0}$.

From the figure, it can be seen that $A_0 \rightarrow A_1$ does not hold, as $m_0 < m_1$.

Similarly, if L_2 is built upon L_1 (Fig. 3), it is obvious from the resulting values that FD $A_0A_1 \rightarrow A_2$ also does not hold. On the other hand, if L_3 is built upon L_1 (Fig. 4), we can see that $m_3 = m_1$, thus, $A_0A_1 \rightarrow A_3$ holds over the dataset. Remark: This can be checked in T as well, as there are no cases where two tuples have the same values for A_0 and A_1, but different ones for A_3.

2.2. Evaluation sequence

For N attributes, there are 2^N possible sequences that could be examined in this way. However, that would result in a lot of redundant evaluations, e.g., sequences $A_0A_1 \rightarrow A_2$ and $A_1A_0 \rightarrow A_2$ are basically the same. Therefore, only a subset of these is needed to be taken.

Figure 5 shows the set containment lattice, a directed graph that shows the order of the building of new layers (where the last attribute is used for the newly built layer). An ordering (panel b) can be set up where each step either builds only one layer (marked red) on top of the one built in the previous step (marked green), or steps back an attribute and builds the new one on a previous one. To produce this series, in this paper, a specially crafted directed graph is created, where each node has a unique ID from 0 to $N-1$, and a directed connection is from each node towards nodes with higher IDs. For example, node #2 is directed towards nodes #3, #4, and #5. On this graph, depth-first search is used, namely, each time the algorithm enters a new node, its layer gets built. Each time the algorithm needs to step back (as there are no more directed connections to move on), the sequence needs to step back too. The results are stored in lattice sequence arrays: L^I and L^B. The former stores which attribute needs to be built in a given step, and the latter, if the next step needs to go back to the previous attribute and use that as a base for the next layer.

As can be seen in Fig. 5b, this results in evaluating the attribute combinations that start with attr. #0, so the sequence of the attributes is rotated $N-1$ more times ($A_0A_1A_2A_3$ first, then $A_1A_2A_3A_0$, $A_2A_3A_0A_1$, and $A_3A_0A_1A_2$), so all ($N\,2^{N-1}$) relevant combinations are made.

2.3. Functional dependency extraction

Finally, with these in mind, the proposed method can be summarized as follows:

- Building the lattice sequence (L^I, L^B), then the static structures (V, C^L, C^R) for each layer.

Experimental results on 7 different benchmark datasets. TABLE I

Dataset	Iris	WBC	Chess	Solar flare	Glass	Abalone	Seismic
N	5	10	7	13	11	9	16
P	150	683	28056	323	214	4177	2584
FDs	4	20	1	9	170	137	374
Lattice size	32	5120	448	53248	11264	2304	524288
Required time [s]	0.003	0.246	0.6	0.75	1.85	47.072	2426.1
Avg. structure size [MB]	0.698	0.45	4.44	0.756	3.56	67.54	18.4
Highest structure size [MB]	0.699	0.569	5.63	0.885	3.56	318.82	22.64
Highest process memory usage [MB]	9	12	29	11	19	629	391

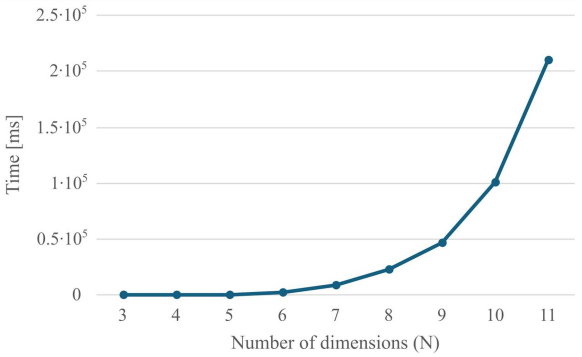

Fig. 6. The dependency extraction time for the different attribute number variations of the abalone dataset.

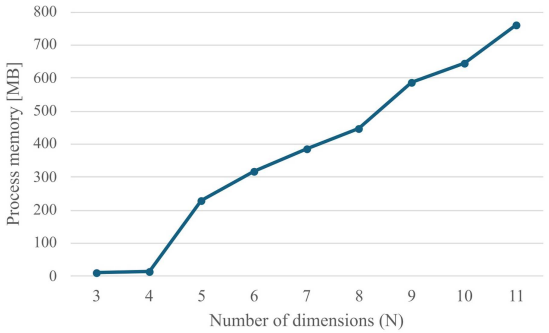

Fig. 7. The highest process memory usage for the different attribute number variations of the abalone dataset.

- Going through the lattice sequence, building a single layer accordingly, and checking if the number of indices in it is the same as the last one. If so, then the attributes involved are added to the FD list.
- Going through the FD list and removing the non-minimal FDs, i.e., the ones with proper subsets that are only FDs regarding the dependent attribute.

2.4. Experimental results

The following experiments were performed using an average PC (Lenovo Legion 7 16ACHg6, AMD Ryzen™9 5900HX CPU, 32GB RAM, NVIDIA GeForce RTX 3080 16GB) using MS Visual Studio Community 2022 and C# .NET framework 4.7.2. This environment has been primarily chosen for its tools for easy development and error analysis, as the goal of the presented experiments is to demonstrate the operability of the SIST FD extractor and analyze how it scales with the dataset size (the number of attributes and tuples).

In the first set of experiments, benchmark datasets (taken from the UCI machine learning repository [11]) have been evaluated using the proposed SIST method: the iris, chess, Wisconsin breast cancer (WBC), abalone, glass identification, solar flare, and seismic bumps datasets. Each dataset has been evaluated 100 times, and the average values are presented in Table I. The proposed method found all FDs that are present in the datasets. The evaluation time is measured (of which, unsurprisingly, the dependency extraction step takes most of the operation time, as processing the lattice takes a number of iterations that is an exponential function of the number of attributes). The memory requirement is also measured, i.e., the average and highest size of the built structure and the highest value of the process memory usage. The latter is taken using the Diagnostic Tool of the VS framework. Remark: The average process memory usage has been typically 30–50% of the highest value.

In the second experiment, the scalability of the attribute number has been examined, for which the abalone dataset has been taken and modified: 6 datasets have been created by choosing only the first 3 to 8 attributes, while 2 more have been made by generating random numbers (real values between 0 and 1). This resulted in 9 datasets with N ranging from 3 to 11, and each has been evaluated 10 times. The resulting average dependency extraction times can be seen in Fig. 6, clearly showing the exponential nature that is inherent in FD extraction. However, Fig. 7 shows that the highest process memory sizes increase linearly with N.

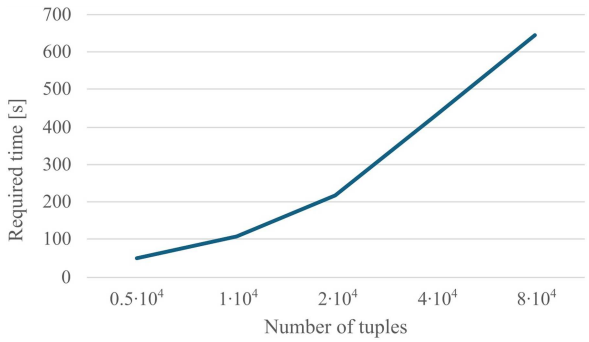

Fig. 8. The dependency extraction time for the different tuple number variations of the abalone dataset.

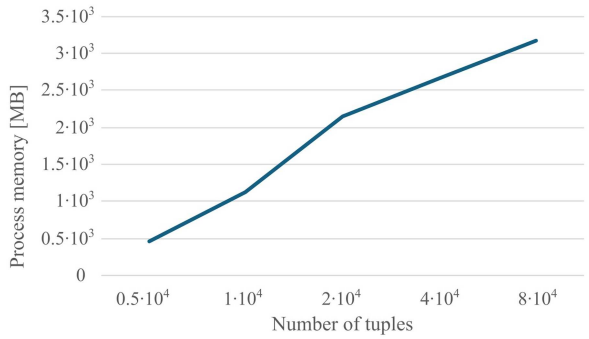

Fig. 9. The highest process memory usage for the different tuple number variations of the abalone dataset.

In the third experiment, the increase in the size of the dataset (i.e., the number of tuples) is examined. For this, new datasets have been generated, once again taking the abalone dataset as a base, but this time, for each new tuple, a value has been chosen from a random row in the abalone dataset (of the corresponding column). This resulted in datasets with 5000, 10000, 20000, 40000, and 80000 tuples. Figures 8 and 9 show that the dependency extraction time and the highest process memory need are both increasing linearly with the number of tuples.

2.5. Computational complexity

The computational and spatial complexity of the proposed SIST method is compared to that of its predecessor method in Table II. The proposed method builds a layer (processing each of the P tuples in a BST with an average size of $\bar{V}$) for each of the $2^N N$ lattice sequence steps. While it is linear considering P, it has an exponential dependence on N. In this regard, the SITs method is faster (φ denotes the number of FDs), though it uses a heuristics that makes it prone to miss some of the FDs, a complete FD analysis (using the presented lattice sequence) would take it $\mathcal{O}(N\,2^{N-1}P)$ to complete. On the other hand, the SIST method produces a much smaller structure, as the average number of unique values ($\bar{V}$) is generally much smaller than the largest size of the value domain ($d_{\max}$) among the attributes.

TABLE II

Comparison between the computational and spatial complexity of the predecessor SITs method and the proposed SIST method.

	Computational complexity	Spatial complexity
SIT	$\mathcal{O}\left(N^3 P\varphi\right)$	$\mathcal{O}\left(N\,P D_{\max}\right)$
SIST	$\mathcal{O}\left(N 2^{N-1} P\,\log_2(\bar{V})\right)$	$\mathcal{O}\left(N\,P\,\bar{V}\right)$

3. Discussion

Overall, the results of the experiments and complexity analysis show that the proposed SIST method works best on datasets with attributes that have somewhat restricted values (e.g., categorical or integer), as real values tend to result in a bloated structure (large but sparsely filled arrays). Its predecessor method had a similar, though even more pronounced problem, namely that whole regions of the value domains of the attributes needed to be stored as arrays. Although the SIST method significantly reduced this with the use of search trees, the structure can still be further optimized to reduce the number of unused array cells in the index array μ.

It is linear in memory need due to the lack of pairwise comparisons, but its exponential dependence on the number of attributes is still a significant downside.

In its current form, the SIST method is basically a "brute force" approach that evaluates all significant attribute combinations and then just removes the non-minimal ones afterward. If the lattice sequence were such that these non-minimal FDs would not be evaluated at all in the first place, this would result in a significantly shorter operation time.

Another possible way to enhance the speed of the SIST method is the application of parallel computing.

A definite advantage of the proposed method is that it is easy to implement, as it does not involve any complex computation or structure.

4. Conclusions

In this paper, a new functional dependency extraction method named Sequential Fuzzy Indexed Search Trees is proposed, which is a specialized

version of the SFISTs classifier. It uses indexing tables and half-balanced binary search trees to build a layered structure using the value combinations of the tuples in the given dataset, and from a trivial indicator in the last layer it can determine if the attribute of the last layer is functionally dependent on the set of attributes used to build the previous layers or not. Its operability is demonstrated using benchmark datasets.

The method is fast in investigating a single FD, but processing all possible FDs results in an exponential computational complexity, so in future work a better approach will be developed.

Furthermore, we will further develop the proposed method to also indicate approximate functional dependencies (which are FDs that hold over the dataset, aside from a small percentage of exceptions), and explore the possibilities of finding conditional functional dependencies as well.

Acknowledgments

This work was supported by the ÚNKP-23-4-II-OE-59 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund. This publication is also the result of the Research & Innovation Operational Programme for the Project: "Support of research and development activities of J. Selye University in the field of Digital Slovakia and creative industry", ITMS code: NFP313010T504, co-funded by the European Regional Development Fund.

References

[1] E.F. Codd, *Commun. ACM* **13**, 377 (1970).

[2] G.N. Paulley, Ph.D. Thesis, University of Waterloo, 2000.

[3] P. Bohannon, W. Fan, F. Geerts, in: *Proc. of the Int. Conf. on Data Engineering (ICDE)*, 2007, p. 746.

[4] M. Le Guilly, J.M. Petit, V.M. Scuturici, in: *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLIV*, Eds. A. Hameurlain, A.M. Tjoa, P. Lamarre, K. Zeitouni, Springer, Berlin 2020 p. 132.

[5] H. Yao, H.J. Hamilton, *Data Min. Knowl. Discov.* **16**, 197 (2008).

[6] J. Liu, J. Li, C. Liu, Y. Chen, *IEEE Trans. Knowl. Data Eng.* **24**, 251 (2012).

[7] A.R. Várkonyi-Kóczy, B. Tusor, J.T. Tóth, in: *Recent Global Research and Education: Technological Challenges. Advances in Intelligent Systems and Computing*, Vol. 519, Eds. R. Jabłoński, R. Szewczyk, Springer, Cham 2017, p. 403.

[8] B. Tusor, J.T. Tóth, A.R. Várkonyi-Kóczy, in: *IEEE 23rd Int. Conf. on Intelligent Engineering Systems*, IEEE, 2019, p. 307.

[9] B. Tusor, J.T. Tóth, A.R. Várkonyi-Kóczy, *Acta Polytech. Hung.* **16**, 65 (2019).

[10] B. Tusor, O. Takáč, Š. Gubo, A.R. Várkonyi-Kóczy, in: *Recent Advances in Technology Research and Education, Lecture Notes in Networks and Systems (Inter-Academia 2023)*, Vol. 939, Eds. Y. Ono, J. Kondoh, Springer, Cham 2024, p. 294.

[11] D. Dua, C. Graff, The UC Irvine Machine Learning Repository, 2024.