SIT-based Functional Dependency Extraction

Balázs Tusor, János T. Tóth, Annamária R. Várkonyi-Kóczy

Department of Mathematics and Informatics, J. Selye University
Elektrárenská cesta 2, 945 01 Komárno, Szlovákia
E-mail: tusorb@ujs.sk; tothj@ujs.sk; koczya@ujs.sk

Abstract: In the digital age, being able to determine the relationships between attributes in datasets is advantageous for many fields of information technology, such as data mining, machine learning, etc. There are inherent rules, functional dependencies that are derived from the nature of the data, of which many are not trivially obvious to the human data manager. This paper presents a new method to extract such relationships. It uses Sequential Indexing Table structures that can implicitly indicate if the values of an attribute are determined by the combination of the values of the attributes preceding it in the layered architecture, i.e. is functionally dependent on those attributes. A new algorithm is given to use this feature to extract functional dependencies, and the performance is analyzed using real-life datasets.

Keywords: functional dependency extraction; data processing; functional dependencies; data mining; sequential indexing tables

1 Introduction

Data management has been an important part of information technology, gaining more and more prominence in the past few decades. Fields such as big data [1] [2] [3], business intelligence [4] [5] and even machine learning [6] [7] [8] [9] profit tremendously from data analysis that helps discover various hidden relationships between the features of data.

There are many functional dependencies (FDs) that inherently derives from the nature of the data. Although many FDs can be recognized intuitively, (e.g. for a persons database: the name, birth date and place are enough to uniquely identify each person), but many of these relationships are often not trivial for human data managers, thus there is a need for a fast, algorithmic extraction of such relationships.

An extensive research has already been put into FD discovery. The three most widely referenced to, classic approaches are TANE [10], FastFDs [11] and Dep-miner [12]. While TANE uses an exhaustive breadth-first search on the data using

a containment lattice to find the functional depedendencies that hold over them, FastFD takes the difference set of the values of the data samples, applies heuristics and a depth-first search to calculate the minimal covering of the created difference sets. Even though they are efficient at finding FDs, most methods in the literature scale badly with the size of the schema and the number of tuples (data samples), which negatively impacts the time they require for operation [12] [13].

Sequential Indexing Tables and Sequential Fuzzy Indexing Tables (SITs and SFITs [14]) have been proposed in order to implement a classifier that uses the bare minimal steps necessary to classify patterns (i.e. find the known pattern (class) that is the most similar to the input values).They are based on the idea of Lookup Tables ([15] LUTs), which store the precalculated value or class label for all possible input value combinations in suitable arrays [16]. Depending on the problem space, this often results in very large, but sparse arrays (with only a small portion of the stored data being useful). SITs are an attempt to reduce the area of the problem space that is in the focus of the classification, thus only storing areas that hold useful information. This is done sequentially, in a layered structure where each layer restricts the problem space by a given value of an attribute, gradually combining them until the search area is reduced to a single point (SITs)or its fuzzy neighborhood (SFITs) in the problem space. The classification performance of SFITs have been thoroughly investigated by the authors in previous works [17] [18] [19] [20] [21].

SITs and SFITs have an implicit property that they can indicate the presence of functional dependencies in the dataset they have been trained with. If the information that an attribute carries (which is supposed to further restrict the problem space by adding it to the combination of the sequence) is already expressed in the combination of the previous attribute values (i.e. it is functionally dependent on them) then it can be detected in the structure.

In this paper, we describe a new functional dependency extraction approach. It builds upon the idea of using SIT structures to detect and subsequently extract the functional dependencies that hold over the dataset schema considering the available data.

The rest of this paper is as follows. In Section 2 the proposed method is described: in Subsection 2.1 a formal definition of functional dependencies is given, then in Subsection 2.2 the architecture of Sequential Fuzzy Indexing Tables is detailed, while in Subsection 2.3 the new functional dependency extraction method is proposed. In Subsection 3.1 the performance of the proposed method is evaluated, while in Subsection 3.2 complexity analysis is given: Subsection 3.2.1 investigates the time complexity, while 3.2.2. the spatial complexity of the proposed approach. Subsection 3.3 describes possible ways for the expansion of the method in future work. Lastly, the conclusions are drawn.

2 Functional Dependency Detection

2.1 Functional Dependencies

Functional dependencies (FDs) are relationships between attributes of a dataset: such a rule states that the value of an attribute is uniquely determined by the values of other attributes. Formally: over a given relation schema (i.e. the set of attributes) $R, A \in R$ and $X \subseteq R$. $X \rightarrow A$ functional dependency is valid in a given relation r (i.e. the dataset) over R:$\forall u, t \in r,\ \forall B \in X$: $if\ t[B] = u[B]$, then $t[A] = u[A]$. For the rest of the paper, the attributes in the left side of a dependency are regarded as the determinant attributes, and the attribute on the right side is regarded as the dependent attribute.

2.2 Dependency Detection with Sequential Indexing Tables

SITs implement an iterative reduction of the problem space, using the values of the input data directly to restrict the area of focus. In the layered structure of SITs each layer corresponds to an attribute. An index array Λ is used in each layer to stores index values that are assigned to each value combination that occurs in the training dataset. The order of the attributes is significant, using two different orderings results in two structures that are different in the sizes of their arrays (though hold basically the same amount of information overall). Each index array has as many columns as the size of the interesting domain of its corresponding attribute (i.e. the range for that attribute in which the training dataset takes values from), and as many rows as the number of index values (so-called *index markers*) that have been assigned in the previous layer (in the first layer, the index array Λ^{L_0} consists of only one row).

Remark #1: Sequential Fuzzy Indexing Tables (SFITs) are the fuzzy extension of SITs, in which an additional fuzzy array is used in each layer in order to handle areas instead of strict integer values (thus adding a certain level of generalization ability to the system, at the price of doubling the size of its structure). However, for classical functional dependency detection and extraction SITs are sufficient as well.

Remark #2: the floating-point values of each tuple are converted into integer format within arbitrary bounds. This is generally done through a suitable linear mapping function:

$$\tilde{X} \cong a \cdot X + b \tag{1}$$

where input value X is scaled with a, and biased with b ($a \epsilon \mathbb{R}$, $b \epsilon \mathbb{N}$) then rounded.

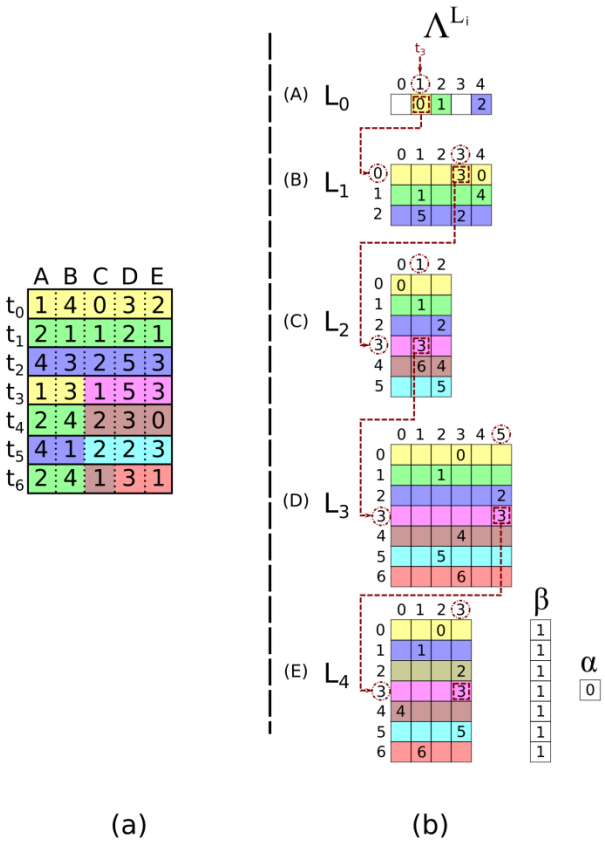

Figure 1

The architecture of Sequential Indexing Tables used for FD detection and extraction

The SIT structure and its usage can be seen in Fig. 1. Let us consider the 7 input tuples $t_0 \ldots t_6$ (a), which have been loaded into the SFIT structure (b). In the first iteration,tuple t_0 = (1,4,0,3,2) is inserted. Since there is no marker in $\Lambda_1^{L_0}$, a new one is set: $\Lambda_1^{L_0} = 0$. This also marks that in the next layer the row #0 is considered. In the second layer $\Lambda_{0,4}^{L_1} = 0$ is set, in the third $\Lambda_{0,0}^{L_2} = 0$, etc. Tuples t_1 and t_2 are inserted similarly. However, for the values of t_3=(1,3,1,5,3) there are already markers in place in the first layer, and thus the insertion process follows the route that the markers have marked (which is highlighted with dashed arrows): it only begins inserting new markers in the second layer (L_1). The input data and the rows of the architecture in the figure are colored accordingly in order to help viewing the logic behind the data structure. Each individual stored pattern or tuple realizes a route from L_0 to L_{N-1}.

In the figure, aside from the first layer, every change of color shows that there is a branching in the 'route of restriction' (of the problem space) that the insertion of the data takes throughout the structure.

In each layer, only a specific row is important for a given pattern (given by the marker in the previous layer). Since each marker leads to a given row in the next layer, a row with only one marker in it can be regarded as a straight path, while if there are more than one markers, then it can be regarded as a junction. In the rest of the paper, the former will be regarded as *singular*, while the latter as *non-singular rows*. Out of the two cases, the latter has more importance, as it introduces additional information to the system (distinguishes tuples based on their attribute value). For classification problems, if the index array of a layer L_i only has singular rows, then it does not contribute to the classification process in any meaningful way, therefore it can be ignored and thus save time. This is because the tuples in the structure until layer L_i are already uniquely distinguished by the combination of the layers (or a subset of the layers) that precede it.Which means that for given values of (at least one of) the previous layers, L_i will only take specific values, which is the definition of functional dependencies. Therefore, if there are only singular rows in the index array of a (non-first) layer, then its attribute is the dependent of a functional dependency, and the determinant attributes are among the attributes belonging to the preceding layers.

This can be used to detect functional dependencies: take an ordering of the attributes such that the investigated attribute is in the last place (N-1), and build the structure. While filling the arrays, in order to avoid the need to examine the whole of the index matrix of the last layer, the numbers of markers for each row are stored in a column vector β. If any of the elements in is raised above one, a variable α is incremented as well, which indicates if there are any non-singular rows. Thus, after building only α is needed to be checked. If α=0, then the attribute is functionally dependent.

2.3 Dependency Extraction with Sequential Indexing Tables

The structure described in Subsection 2.2 can also be used to determine which layers are the determinants of the functional dependencies. The base idea is to create an ordering in which the last layer (N-1) corresponds to the attribute that is needed to be examined as the dependent attribute (i). If the index array of the last layer is singular (i.e. all of its rows are singular, α=0), then the analysis can commence (otherwise, the attribute is skipped).

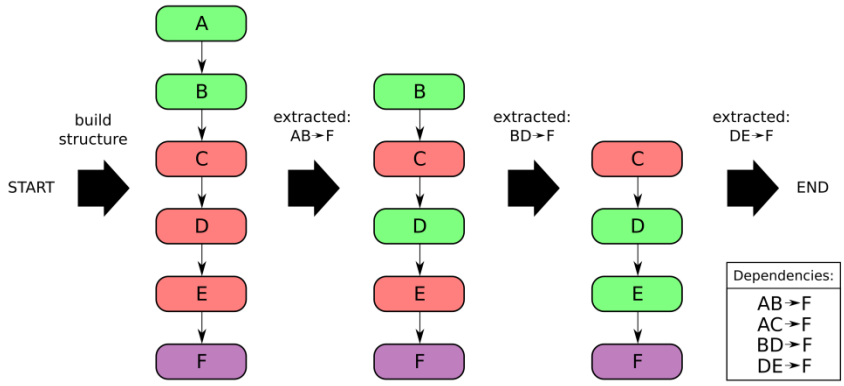

Figure 2
The operation of the FD extractor on a forward order

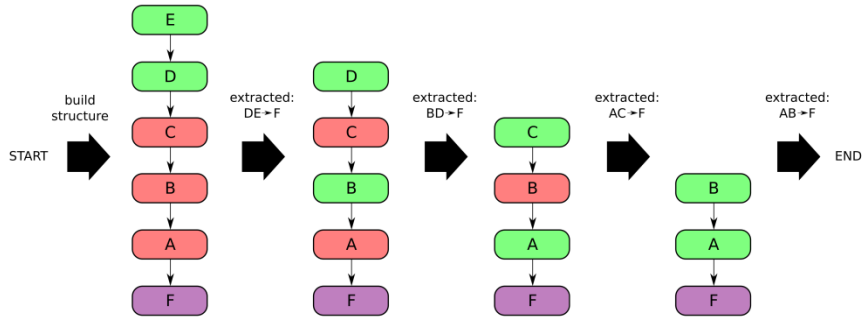

Figure 3
The operation of the FD extractor on a reverse order

The analysis is done in iterations. The structure is built again, but this time without L_{N-2}(the layer that directly precedes the last one). If α=0 in the newly rebuilt structure, then attribute of L_{N-2} is not significant in the investigated dependency, it can be ignored in the next steps and move on to the layer before it. If α>0, then the preceding layers are no longer sufficient to uniquely identify each stored tuples, information is lost if the layer is skipped. Thus, it is restored and noted as being a part of the determinant. This is done until the first layer is reached and examined.

This algorithm, however, only returns the FD that has an attr. in the highest layer in the topology. This is illustrated in Fig. 2, where a simplified topology stores a dataset, with a schema of R=(A, B, C, D, E, F) where AB→F, AC→F, BD→F and DE→F functional dependencies hold. In the first iteration, E is ignored and proven to be unnecessary, as A and B determines the values of F on their own (and are thus marked with green). D and C are also insignificant in this regard (marked with red). At the end of the analysis, the system returns with AB→F. In order to

extract the rest of the FDs, the evaluation needs to be restricted: before the next round, attribute A is removed, as it is the highest in the topology among the attributes of the found FD. In the next round, BC→F is found, then after B is removed, DE→F. However, as it can be seen, this method in itself could not find AC→F, because A was removed prematurely (this is always the case when multiple FDs have the same common attribute, but is high in the topology so it gets removed before all of the FDs are discovered). This can be amended by running the algorithm a second time, but for a *quasi-reversed* order (Fig. 3): reversing the order of the potential determinant attributes but leaving the dependent attribute at the end. The output of the system is the union of the two sets of FDs discovered in each phase (normal and reversed order).

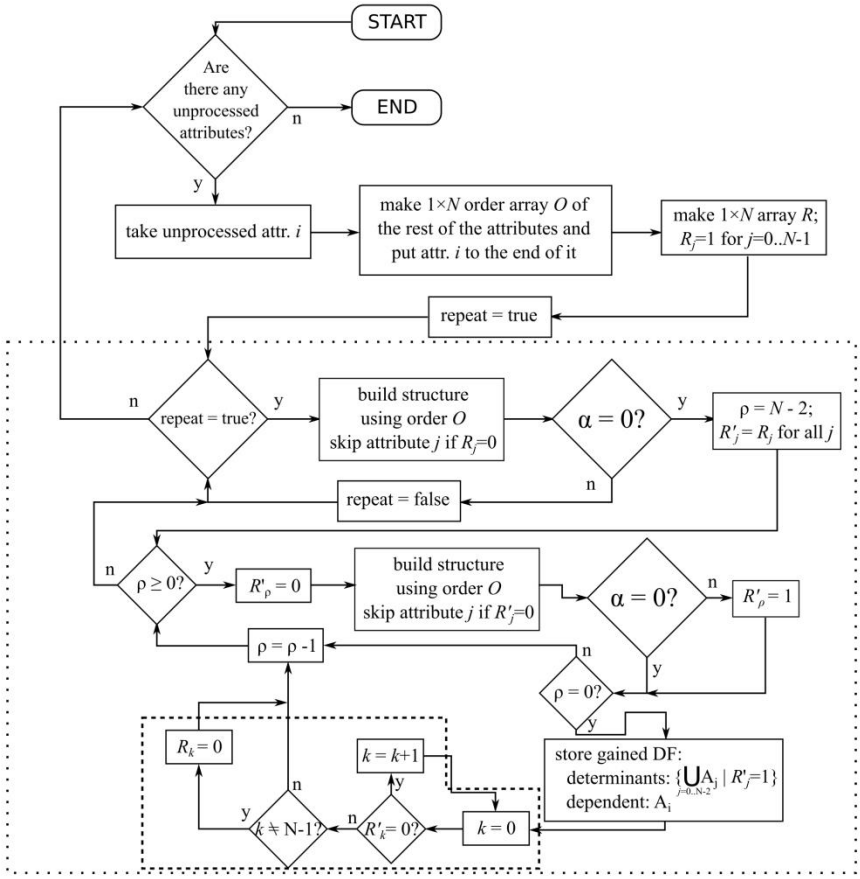

Figure 4
The algorithm of the FD extractor

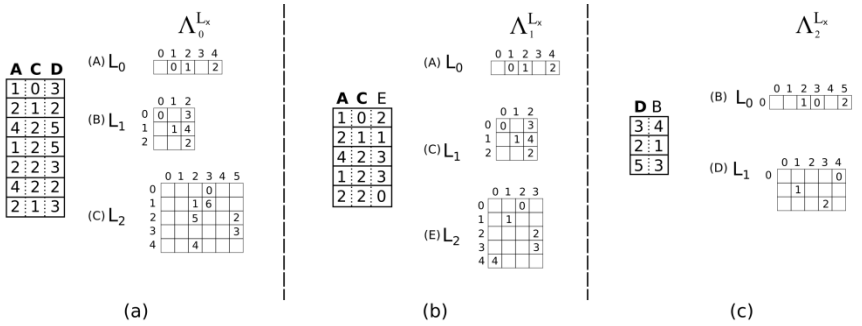

Figure 5
Divided schemas (the key attributes are marked with bold)

Fig. 4 summarizes the algorithm described above, with more details. The ordering of the attributes is stored in 1x*N* array *O*, while the emission of certain attributes or layers is done through 1x*N* array *R*, that stores which attributes are *regarded* in the given iteration. It is used to remove the top attribute of the previously found FD. On the other hand, the temporal removal of layers (from the bottom of the structure) is stored in *R'*. For each examined attribute, first the auxiliary structures are initialized (*O* and *R*), then the FD extraction step (marked by a bounding box with doted borders). In the extraction step, the SIT structure is built using ordering *O* regarding the attributes marked by *R*. If the index array of the last attribute is singular $\alpha>0$ then there are no more FDs regarding the attr. *i*, the next attribute is regarded. Otherwise, *R* is copied into *R'*, the appropriate layer from the bottom (but above L_i) is marked as disregarded ($R'_\rho=0$) and a new SIT structure is built using order *O* and *R'*. If $\alpha>0$, then reinstate the layer ($R'_\rho=1$) and go on, until the first layer is reached ($\rho=0$), at which point *R'* marks the attribute of the determinants of the discovered FD. Lastly, the first attribute of the FD is determined (marked by a smaller boxed area in the figure with dashed border), and *R* is set accordingly.

An interesting (and valuable) consequence of the algorithm is that it always returns the *minimal* FDs, therefore the left side of the relation does not need to be pruned to get the smallest determinant subset that determines the dependent attribute on the right side. The significance of functional dependencies is illustrated in Fig. 6. Examining the same dataset that was used in Fig. 1, the method described above returned the following FDs: B→D, D→B, E→C, AC→E and CD→E.Therefore, the 5-attribute schema in Fig. 1 can be broken up into 3 schemas: (a) ACD, (b) ACE and (c) DB. From a database management viewpoint, this is very valuable as it helps in keeping data consistency, makes modifications easier (e.g. if a value of D changes, then only one element of schema ACD is needed to be changed). Furthermore, if the schemas are stored in SITs, this separation also reduces its size: the structure of ABCDE uses 108 elements, while the 3 combined schemas need only 99 elements (which may not seem like a big reduction, but for larger scales the gap between the numbers is also larger).

3 Performance Evaluation

3.1 Experimental Results

The proposed functional dependency extraction method had been tested on 4 real-life databases from the UCI data repository [23] on using an average PC (Intel® Core™ i5-4590 CPU @ 3.30 GHz, 16 GB RAM).

In the first set of experiments, the Abalone dataset is analyzed with the proposed method, and the effects of the scaling coefficient is examined. The dataset consists of 4177 training samples with 9 attributes. Its original purpose is using the first 8 features (gender, length, diameter, height, whole weight, shucked weight, viscera weight and shell weight) in order to determine the age of the animal (which is measured by the rings of its shell), in hopes that a method can be found that does not necessarily involve the expiration of the abalone (as the counting of the rings requires sawing the shell in half, which has a detrimental effect on the health of the creature.)

Table 1
Number of FDs found for each feature as dependent attribute of the Abalone dataset

	Scaling Factor		
Attribute	**200**	**500**	**1000**
Gender	3	6	7
Length	2	3	5
Diameter	2	4	5
Height	0	3	5
Whole_weight	3	2	4
Shucked_weight	1	5	5
Viscera_weight	3	2	4
Shell_weight	1	4	6
Rings	0	4	7
Number of all FDs found	15	33	48

Table 2
Average time requirements for FD extraction on the Abalone dataset based on the scaling factor

	Scaling Factor		
	200	**500**	**1000**
Average time of FD extraction	4.323 s	13.543 s	23.806 s
Average time for building a single SIT structure	0.024 s	0.048 s	0.0545 s

Table 3
Average time requirements for FD extraction on the Abalone, Glass, Wisconsin breast cancer and Iris datasets and the number of extracted FDs

	Number of attributes	**Number of tupples**	**Average time of a full analysis**	**1 build time**	**found no. Of FDs**	**Number of FDs of the last attr.**
Abalone (*a*=1000)	9	4177	23.806 s	0.0545 s	48	7
Glass	10	214	1.13 s	0.04 s	70	8
Wisconsin breast cancer	10	683	0.074 s	0.002 s	6	6
Iris	5	150	0.01 s	<0.001 s	2	2

While the gender and the number of the rings are integer numbers, the rest of the features are floating point numbers that are determined to the 3rdor 4th fraction digit. This requires a suitable scaling factor (a_i) that transforms the data into a suitable integer domain. Three different scaling factors are investigated: 200, 500 and 1000. Table 1 shows the results, listing the FDs for each attribute. Interestingly, the larger the scaling factor, the more FDs are discovered that hold on the dataset (15, 33 and 48, respectively). This is due to more details and thus information is lost if the scaling factor is too low. On the other hand, the necessary time is much larger (4.323, 13.543 and 23.806 seconds, respectively), due to the larger sizes of the structure needed to be maintained and the large number of iterations. The average building time of a single SIT structure on the other hand took only 0.024, 0.048 and 0.054 seconds, respectively (Table 2). As for the original purpose of the dataset, the discovered FDs show that the number of rings cannot be directly determined from only the features that does not require the demise of the animal (the closest one features length, diameter, whole weight and shucked weight as the determinants, of which the last one causes the demise of the abalone).

The performance of the FD extractor has been evaluated on 3 other datasets as well: the Iris, the Glass and the Wisconsin breast cancer datasets. The Iris dataset (which is about finding the species of iris plants: Iris Setosa, Iris Versicolor and Iris Virginica) consists of 150 tuples and 5 integer valued attributes. It takes about 10 ms to find the two FDs that holds on the schema. Table 3 summarizes the results. The Wisconsin breast cancer (WBC, determining the nature of mammary tumors) dataset takes slightly more time (74 ms) to process the 683 tuples with 10 integer valued attributes, finding 6 FDs. Finally, in the Glass dataset (determining

the type of glass from its chemical components) has been processed with 214 tuples of 10 floating point features (with suitable scaling factors), finding 70 FDs in 1.13 seconds, of which 8 FDs had the type of glass as the dependent attribute. Interestingly, for the WBC and Iris datasets only the classification target (type of plant and tumor) are determined by the other attributes, but none of the other attributes. The Solar flare data set has also been processed, but no FDs have been found.

3.2 Complexity Analysis

3.2.1 Time Complexity

The computational complexity of FastFDs, TANE ([10][11]) and the SIT-based FD detector and extractor is compared in Table 4. Let *R* denote the schema (the set of attributes) and *r*mark the set of values in the dataset, and thus $|R|$ and $|r|$ the number of each set, respectively. If only FD detection is the goal, then the proposed method only requires building a SIT structure (each attribute value is used for one layer only $O(|r|\cdot|R|)$), for each attribute:

$$O(|r|\cdot|R|^2) \tag{2}$$

FD detection is sufficient if the goal is to enhance the speed of a classifier: since the dependent attributes do not contribute additional information into the classification process, skipping them can speed up most classifiers.

If the detected FDs are needed to be extracted as well, then the detection step is needed to be repeated for each attribute, and for each FDs in the dataset:

$$O(\phi\cdot|r|\cdot|R|^3) \tag{3}$$

where ϕ is the number of FDs on a schema considering the given data. The computational complexity of the proposed method is polynomial with respect to the size of the schema ($|R|$), and is linear in the number of tuples ($|r|$), as opposed to the other two methods.

Table 4
Time complexity analysis for Tane, FastFDs and the SIT-based FD detector and extractor

Method	Computational Complexity
TANE	$O(2^{\|R\|}\cdot(\|r\|+\|R\|^{2.5}))$
FastFDs	$\sim O(\|R\|\cdot\|r\|^2+\|R\|\cdot\|r\|^2\log(\|R\|\cdot\|r\|^2)$
Dependency Detector SIT	$O(\|r\|\cdot\|R\|^2)$
Dependency Extractor SIT	$O(\phi\cdot\|r\|\cdot\|R\|^3)$

3.2.2 Spatial Complexity

Although the proposed method is less complex computation-wise than the other described methods, it is important to examine its memory requirement as well. The size of the main structure (the index matrices in each layer) of the FD extractor only depends directly on the number of attributes and the size of the domains of each attribute, while it is much less influenced by the number of tuples. On the other hand, the number of tuples can be used as an upper bound to the number of rows (ρ) in the index matrix of each layer:

$$\rho_i = \begin{cases} 1 & , \quad i = 0 \\ \gamma_{i-1} \leq \min(D_{i-1} \cdot \rho_{i-1}, |r|), & i > 0 \end{cases} \tag{4}$$

where ρ_i is the number of rows of the index array in layer i and γ_{i-1} is the number of the index markers in the previous layer. The latter number is typically only known after counting the unique value combinations in the previous layer. Thus, the building of the structure can be done in two ways: going through all of the input data tuples in each layer to get the exact amount of unique value combinations (and thus, γ_i), or construct a structure that is expectedly much larger than necessary using the known upper bounds (from the domain size and number of rows of the previous layer ($D_{i-1} \cdot \rho_{i-1}$), or the number of input tuples ($|r|$)); then go through the input data once, note the γ values for each layer and rebuild the structure with the accurate size. The latter obviously requires less time to do (as the input data is only processed twice), at the cost of temporarily using more memory.

The size of the whole structure is ideally:

$$S = \sum_0^{|R|-1} \rho_i \cdot D_i + \rho_{|R|-1} + 1 \tag{5}$$

Table 5
Spatial complexity analysis for Tane, FastFDs and the SIT-based FD extractor

Method	Spatial Complexity
TANE	$O\left(\frac{(\|R\| + \|r\|) \cdot 2^{\|R\|}}{\sqrt{\|R\|}}\right)$
FastFDs	$O\left(\|R\| \cdot \frac{\|r\| \cdot (\|r\| - 1)}{2}\right)$
Dependency Extractor SIT	$O(\|R\| \cdot \|r\| \cdot D_{max})$

For most cases, it is likely that the lowest layer in the structure has |r| number of rows (unless there is a significant number of redundant tuples), so using |r| as the upper bounds for the row numbers for the worst-case scenario, the size of the structure can be estimated:

$$S = \sum_0^{|R|-1} |r| \cdot D_i + |r| + 1 \tag{6}$$

Thus, the spatial complexity of the SIT-based FD detector in the worst-case scenario:

$$O(|R| \cdot |r| \cdot D_{max}) \tag{7}$$

The spatial complexity of FastFD, TANE ([10] [11]) and the SIT-based FD detector and extractor is compared in Table 5. The complexity of TANE is exponential, though it can be mitigated by keeping parts of the data on the hard drive, which in turn slows the operation as reading and writing from hard drives is significantly slower than memory operations. FastFD uses significantly less memory, though it is still quadratic in the function of |r|.

The scaling of the input tuple values during the linear mapping step is also very important, considering the spatial complexity of the system. Although higher scaling factor values increase the chance of finding functional dependencies, they also directly affect the size of the domain of their attributes in question, and subsequently, the size of the SIT structure. Fig. 6 depicts how the size of the structure changes with the scaling factor (a_i) using the Abalone data set. The dataset has 9 attributes, of which 7 have floating point values and 2 have integer type values (thus the latter ones are not scaled). The 7 attributes are scaled using 10 different values (from100 to 1000), with the same factor in each step (e.g. a_0=a_2=…=a_6=100).

In the figure, the overall sizes of the main structure (the sum of the sizes of the index matrices, i.e. how many numbers are needed to be stored in them) are compared. As it can be seen, the scaling of the amount of stored data is more or less linear. The smallest examined scaling factor (a_i=100) yieldsa structure with the size of 1940403 elements, while the largest (a_i=1000)results in 24523818 elements. To put this into perspective, in order to store the latter amount of array elements using short data type (2 bytes), the structure takes ~46.7 MB to store.

Remark: *short* data type can only be used if the number of indexes in any layer (which in the last layer is the number of unique input tuples) does not exceed the largest number that can be stored in a *short* type variable (65535). For the Abalone data set this is true (since the number of input tuples is 4177), but for larger data sets integer type variables (4 bytes) are needed instead, doubling the memory requirement of the system.

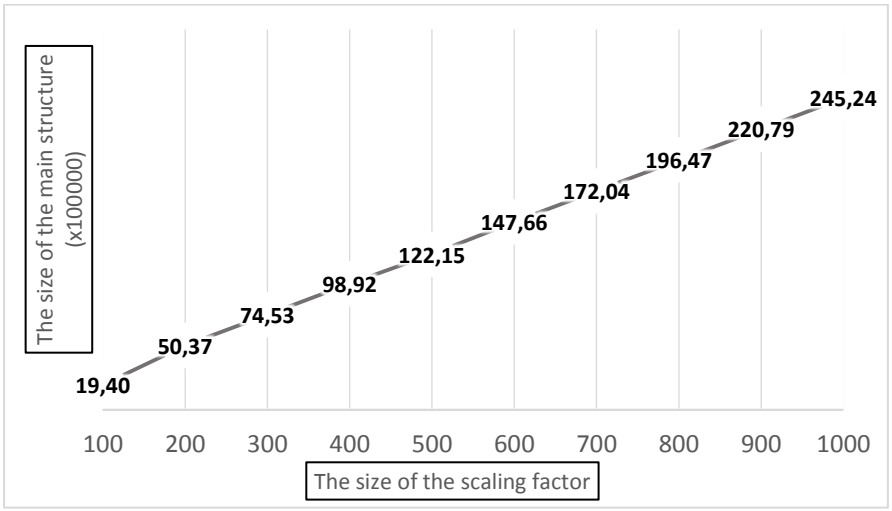

Figure 6
The change of the size of the structure (the amount of array elements stored) in the function of the scaling factorusing the Abalone data set

3.3 Future Extensions

Finding clear rules (FDs) in real-life datasets is often hard, if not downright impossible, especially when the amount of the data is very large (raising the possibility of errors, noise or samples that are truly exceptions of the rule). For such cases *Approximate Functional Dependencies* (AFDs) were defined as FDs that hold on a sufficiently large subset of the available tuples. They can be regarded as general rules that have some exceptions to them as well. Being able to extract these rules could also be very advantageous because with them it is still possible to make predictions towards the values of given attributes (e.g. in case of the Solar flare dataset in which no clear FDs can be found, or the Abalone dataset in which the age of the animal could still be predicted from the features that do not require the its destruction). The SIT architecture can already detect the possibility of an attribute being the dependent in an AFD, simply by examining how many of the tuples end in singular and non-singular rows in the index array of the last layer. If their number is sufficient, then ignoring the rest of the tuples (that end in non-singular rows), the proposed method can be used to detect the FDs that hold on them, and thus extract them as AFDs.

The main disadvantage of the proposed method is that its usage is limited in both the number of attributes and the domain in which the attributes can take values from. These directly determine the size of the structure, which can be too large to manage with the average modern computers. The structure, however, can be changed to omit the non-interesting areas (e.g. the empty elements seen in Fig. 1)

and only store the known values. However, this will increase the complexity of the implementation of method and require parallel computing to keep the management of the arrays fast. On the plus side, this can also make it possible to process the tuples in batches, instead of one by one. This upgrade would make it possible for the proposed method to be applied in Big Data applications as well.

Conclusion

In this paper, we describe a new functional dependency extraction approach. It builds upon the idea of using Sequential Indexing Table structures to detect and subsequently extract the functional dependencies that hold over the dataset schema considering the available data.

The proposed method is shown to be able to extract FDs quickly (its computational complexity is a linear function of the number of input tuples), however, at the cost of larger memory usage. The size of the structure is primarily influenced by the number of attributes and the size of the domain they can get values from. The latter can be restricted with suitable scaling factors, although in return of potential loss of information and thus less reliable operation.

In future work, the size problem of the structure will be amended, and the method will be extended to be able to find Approximate Functional Dependencies as well (by disregarding tuples that count as exceptions, if their number is low). Furthermore, parallel programming techniques will be added into the method in order to enhance its speed by processing the input data samples in batches instead of one by one.

Acknowledgement

This publication was created due to support 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] M. Jocic, E. Pap, A. Szakál, D. Obradovic, Z. Konjovic, "Managing Big Data Using Fuzzy Sets by Directed Graph Node Similarity," Acta Polytechnica Hungarica, Vol. 14, No. 2, 2017, pp. 183-200

[2] R. Spir, K. Mikula, N. Peyrieras "Parallelization and validation of algorithms for Zebrafish cell lineage tree reconstruction from big 4D image data," Acta Polytechnica Hungarica, Vol. 14, No. 5, 2017, pp. 65-84

[3] A. Vukmirović, Z. Rajnai, M. Radojičić, J. Vukmirović, M. J. Milenković, "Infrastructural Model for the Healthcare System based on Emerging Technologies," Acta Polytechnica Hun., Vol. 15, No. 2, 2018, pp. 33-48

[4] T. Szántai, E. Kovács, A. Egri, "Inventory Control in Sales Periods," Acta Polytechnica Hungarica, Vol. 15, No. 1, 2018, pp. 87-104

[5] V. Farrokhi, L. Pokorádi, S. Bouini, "The Identification of Readiness in Implementating Business Intelligence Projects by Combining Interpretive Structural Modeling with Graph Theory and Matrix Approach," Acta Polytechnica Hungarica, Vol. 15, No. 2, 2018

[6] J. Parra, O. Fuentes, E. Anthony, V. Kreinovich, "Use of Machine Learning to Analyze and – Hopefully – Predict Volcano Activity," Acta Polytechnica Hungarica, Vol. 14, No. 3, 2017, pp. 209-221

[7] D. Marček, M. Rojček, "The Category Proliferation Problem in ART Neural Networks," Acta Polytechnica Hungarica, Vol. 14, No. 5, 2017, pp. 49-63

[8] N. Ádám, A. Baláž, E. Pietriková, E. Chovancová, P. Feciľak, "The Impact of Data Representationson Hardware Based MLP Network Implementation," Acta Polytechnica Hungarica, Vol. 15, No. 2, 2018, pp. 69-88

[9] L. Nyulászi, R. Andoga, P. Butka, L. Főző, R. Kovacs, T. Moravec, "Fault Detection and Isolation of an Aircraft Turbojet Engine Using a Multi-Sensor Network and Multiple Model Approach," Acta Polytechnica Hungarica, Vol. 15, No. 2, 2018, pp. 198-209

[10] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," The Comp. Journ., Vol. 42, No. 2, 1999, pp. 100-111

[11] C. Wyss, C. Giannella, and E. Robertson, "FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances," In Proc. of the Int. Conf. of Data Warehousing and Knowledge Discovery (DaWaK), 2001, pp. 101-110

[12] T. Papenbrock et.al.,"Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms," Proceedings of the VLDB Endowment, Vol. 8, No. 10, 2015, pp. 1082-1093

[13] N. Asghar, A. Ghenai, "Automatic Discovery of Functional Dependencies and Conditional Functional Dependencies: A Comparative Study," 2015

[14] A. R. Várkonyi-Kóczy, B. Tusor, and J. T. Tóth, "A Multi-Attribute Classification Method to Solve the Problem of Dimensionality," in Proc. of the 15th Int. Conf. on Global Research and Education in Intelligent Systems (Interacademia'2016), Warsaw, Poland, Sept. 26-28, 2016, pp. PS39-1 – PS39-6

[15] Maher, David. W. J. and John F. Makowski. "Literary Evidence for Roman Arithmetic with Fractions", Classical Philology, Vol. 96, No. 4, 2001, pp. 376-399

[16] B. D. Zarit, B. J. Super, F. K. H. Quek, "Comparison of five color models in skin pixel classification," in Proc. of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, Corfu, Greece, Sep. 26-27, 1999, pp. 58-63

[17] B. Tusor, A. R. Várkonyi-Kóczy, J. T. Tóth, "Active Problem Workspace Reduction with a Fast Fuzzy Classifier for Real-Time Applications," IEEE International Conference on Systems, Man, and Cybernetics, Budapest, Hungary, October 9-12, 2016, pp. 4423-4428, ISBN: 978-1-5090-1819-2

[18] A. R. Várkonyi-Kóczy, B. Tusor, J. Tóth, "Robust Variable Length Data Classification with Extended Sequential Fuzzy Indexing Tables," 2017 IEEE Int. Instr. and Meas. Tech. Conf. (I2MTC) 22-25 May, 2017, Torino, Italy, pp. 1881-1886

[19] B. Tusor, A. R. Várkonyi-Kóczy and J. T. Tóth, "A Fuzzy Data Structure for Variable Length Data and Missing Value Classification," 16th International Conference on Global Research and Education, 25-28 Sept. 2017, Iasi, Romania, pp. O.21-1 – O.21-6

[20] B. Tusor, G. Simon-Nagy, A. R. Várkonyi-Kóczy and J. T. Tóth, "Personalized Dietary Assistant - An Intelligent Space Application," 21st IEEE Int. Conference on Intelligent Engineering Systems (INES 2017), Larnaca, Cyprus, 20-23 October, 2017, pp. 27-32

[21] B. Tusor, A. R. Várkonyi-Kóczy, J. Bukor, "An ISpace-based Dietary Advisor," 2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA) 11-14 June, 2018, Rome, Italy, pp.1-6

[22] B. Kamiński, M. Jakubczyk, P. Szufel, "A framework for sensitivity analysis of decision trees". Central European Journal of Operations Research, Vol. 26, No. 1, 2017, pp. 135-159

[23] D. Duaand C. Graff, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2019