https://doi.org/10.36007/5093.2024.309

STRICT AND APPROXIMATE FUNCTIONAL DEPENDENCY EXTRACTION WITH SEQUENTIAL INDEXING TABLES-BASED SEARCH TREES

Balázs TUSOR [1] – Ondrej TAKÁČ [2] – Annamária R. VÁRKONYI-KÓCZY [3] – Štefan GUBO [4]

ABSTRACT

Data relation analysis has been a very important field of data science for the past few decades, in which the goal is to discover relationships between data attributes. Functional dependencies are among the most basic of such relations, defining a strict determination between the values of the attribute set on the determinant side and the values of the attribute on the dependent side of the relation. Approximate functional dependencies allow a certain amount of deviation from strict dependencies, and thus, can indicate relationships that are true for a large extent, with only a few exemptions. In this paper, a new method is presented for the extraction of strict and approximate dependencies, which applies the same base idea as the SFIT, but with the combination of binary search trees to reduce the required memory without sacrificing much of the operational speed.

KEYWORDS

functional dependency extraction, approximate functional dependency extraction, indexing tables

INTRODUCTION

Data relation analysis has been a very important field of data science for the past few decades, in which the goal is to discover relationships between data attributes. Functional dependencies (FDs) are among the most basic of such relations, defining a strict determination between the values of the attribute set on the determinant side and the values of the attribute on the dependent side of the relation. This is not only helpful in optimizing database systems, but it is also often used in data mining, in order to find hidden causations (or at least, correlations) between the attributes. Approximate functional dependencies (AFDs) allow a certain amount of deviation from strict dependencies, and thus, can indicate relationships that are true for a large extent, with only a few exemptions.

In previous work, the authors have developed a classifier called Sequential Fuzzy Indexing Tables (SFIT) (Várkonyi-Kóczy et al., 2016) that can be used to find strict and approximate FDs very quickly, exploiting the high operational speed of lookup tables. As a trade-off, the SFIT classifier uses a significant amount of memory that scales exponentially

[1] Balázs Tusor, J. Selye University, Faculty of Economics and Informatics, Department of Informatics, tusorb@ujs.sk
[2] Ondrej Takáč, J. Selye University, Faculty of Economics and Informatics, Department of Informatics, takaco@ujs.sk
[3] Annamária R. Várkonyi-Kóczy, J. Selye University, Faculty of Economics and Informatics, Department of Informatics, koczya@ujs.sk
[4] Štefan Gubo, J. Selye University, Faculty of Economics and Informatics, Department of Informatics, gubos@ujs.sk

with the number of attributes. However, one interesting inherent property of the SFIT classifier is that it can indicate the presence of FDs in its structure. Using this property, the so-called Sequential Indexing Tables (SIT) (Tusor et al., 2019a and 2019b) method was created, and further developed to indicate AFDs as well (Tusor et al., 2019c). The method was shown to be able to find FDs and AFDs (in the following: A/FDs), but also had the disadvantageous memory requirement of the base classifier.

In this paper, a new method is presented for the extraction of strict and approximate dependencies, which applies the same base idea as the SFIT, but with the combination of binary search trees to reduce the required memory without sacrificing much of the operational speed.

The rest of the paper is as follows. After a brief literature review and research methodology description, the proposed method is described in detail. Its performance is demonstrated on benchmark datasets, and its computational and spatial complexities are analysed. Finally, the paper is concluded, and future work is described.

LITERATURE REVIEW

A functional dependency is a constraint between two attribute sets in a relation. Let R be a relation (i.e., the full set of the attributes in a given dataset), $X \subseteq R$ and $Y \in R$. Y is functionally dependent on X ($X \rightarrow Y$), if $Y \notin X$ and there are no two tuples in the dataset where the attribute values of X are the same, but the value of Y is different. An approximate functional dependency is an FD that almost holds, i.e., is supported by a sufficient ratio of the tuples in the dataset.

The main problem is that the computational complexity of finding all A/FDs in a given dataset with N attributes and P tuples is $O(P^2 \cdot N \cdot N \cdot 2^N)$ (Liu et al., 2010), comparing each sample to all others (hence the quadratic dependence on the number of tuples), for each of the N attributes, and all of the $N \cdot 2^N$ attribute combinations. Most methods in literature use different heuristics and techniques to circumvent the exponential factor, e.g. *TANE* (Huhtala et al., 1999) uses a partitioning approach, *FastFD* (Wyss et al., 2001) applies a depth-first heuristics-driven search to find the A/FDs in a dataset, while Pyro (Kruse and Naumann, 2018) uses a separate-and-conquer discovery strategy.

The main feature of SITs, as mentioned in the introduction, is that they can indicate the presence of A/FDs in the datasets intrinsically from their trained structure, without the need for pairwise tuple comparisons, so the method is only dependent linearly on P. It also applies heuristics in order to lower the number of examined attribute combinations (by ignoring the non-minimal FDs), which works well on categorical and integer valued data, but tests showed that for real valued data it only found most of the A/FDs.

RESEARCH METHODOLOGY

The primary goal of this preliminary research is investigating if the feasibility of an A/FD extraction method that uses the base idea of the SIT method, but realizes it in a much more compact structure that can handle real valued data easily as well. The other goal is to investigate how its computational and spatial complexity scales with the number of attributes and tuples (without applying any significant heuristics yet). For the implementation of the prototype of the method, MS Visual Studio Community 2022 has been used and C# language (.NET framework 4.7.2.), as its built-in debugging tools making the development process faster.

STRICT FUNCTIONAL DEPENDENCY INDICATION

As mentioned above, Sequential Indexed Search Trees (SISTs) indicate the presence of strict FDs implicitly in their structure, so there is no need for pairwise tuple comparisons. They have a layered architecture, in which each layer corresponds to a different attribute. The first layer divides the set of input tuples into subsets based on the attribute values corresponding to that of the layer, then each subsequent layer further divides the subsets gained in the previous layer (so the layers are *built upon* each other). For each layer L_i, the values the given attribute can take are stored in a semi-balanced binary search tree (i.e., a binary search tree where the difference in the length of each route from the root node to a leaf node is either 0 or 1). The BST is implemented through 3 arrays:

- Value array V that stores the value of each node;
- Left child index C^L that stores the index of the left-hand child of each node;
- Right child index C^R that stores the index of the right-hand child, similarly.

The index of the root node of the tree is also stored (r_i). The nodes are sorted into an ascending order, so they can be easily arranged into a semi-balanced binary tree.

The subset indices in layer i are stored in index array μ (with the size of $S^V \times m_j$, where S^V is the number of nodes in layer i, and j is the number of subset indices in the layer that layer i is built upon). For the very first layer $m_j = 1$. The total number of subset indices in layer i is stored in m_i. An example can be seen in Figure 1, where the first layer (with A_0) is built using the values of dataset T.

In the following layer (i'), the number of subset indices indicate the presence of FDs: if $m_i = m_{i'}$, then layer i' did not divide the subsets (gained in the previous layer i) any further, the values of attribute i' are not in contradiction with the value combinations from the previous layers and thus, an FD is present. Figure 2 shows an example of a layer built upon A_0, using A_1. As it can be seen, since $m_0 \neq m_1$, there is no FD between their attributes (which can be observed in Figure 1(a) as well, e.g., t_0 and t_3 has the same value for A_0, but different values for A_1). Building another layer upon A_1 using A_2 yields similar results (Figure 3), however, doing the same (i.e., building a layer upon A_1) using A_3 (Figure 4) then we can see that $m_3 = m_1$, thus, there is a strict FD between the attributes of the preceding layers and the newly built layer: $A_0A_1 \rightarrow A_3$.

Remark: As this example shows, the building order does not need to strictly follow the original sequence of the attributes, a layer of any of the attributes can be built upon any other layer.

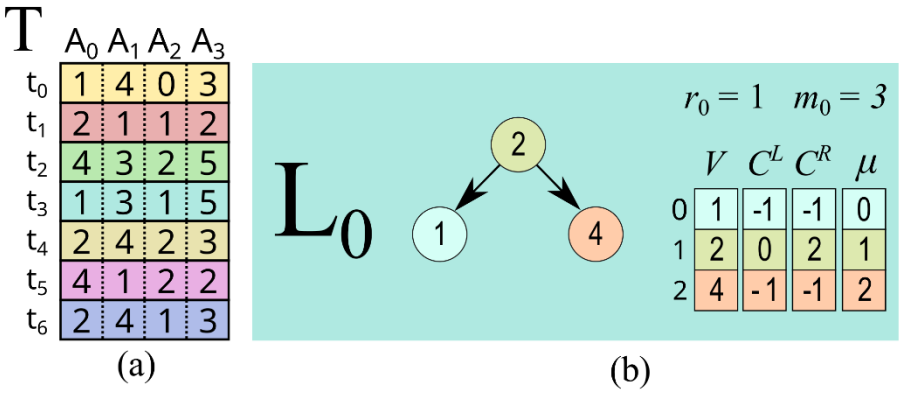

Figure 1: (a) an example for a simple dataset with 4 attributes, and (b) the binary search tree representation of the values of A_0, as well as the index array representation of the tree and the subset IDs.

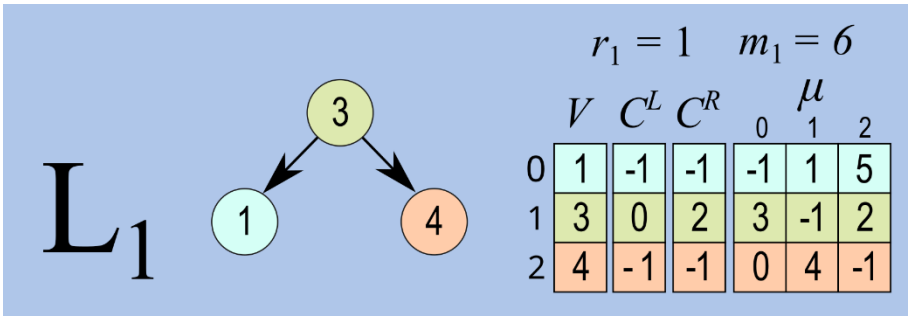

Figure 2: The second layer L_1 that was built on the first layer (L_0), using attribute A_1.

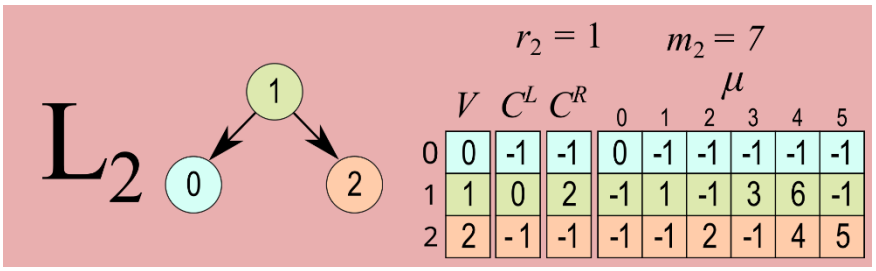

Figure 3: The third layer L_2 that was built on the second layer (L_1), using attribute A_2.

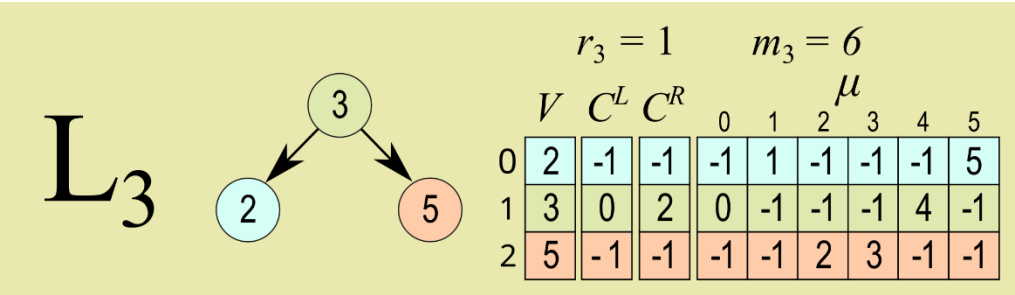

Figure 4: The fourth Layer L_3 that was built on the second layer (L_1), using A_3.

APPROXIMATE FUNCTIONAL DEPENDENCY INDICATION

In order to discover AFDs, the base structure is extended with two arrays. Firstly, the number of samples in each subset are accounted for in array ρ (that has the same size as μ), and the highest number in each column in ρ is stored in 1D array s. After a layer is fully built (i.e., all the tuples have been processed), the sum of the values of s indicates how many tuples support the currently examined relation between the attributes of the previous layers and the attribute of the last layer, and thus, the support can be calculated for layer i (built upon layer j) as:

$$support_i = \frac{\sum_{k=0}^{m_j} s_k}{P}. \quad (1)$$

Figure 5 shows an example for the additional structures for layers L_0 and L_1. As it can be seen, the support for the AFD $A_0 \rightarrow A_1$ is 4 out of 7 (57%). If L_2 is built upon L_1 (using A_2), the resulting support value for $A_0A_1 \rightarrow A_2$ is 6 out of 7 (85.7%).

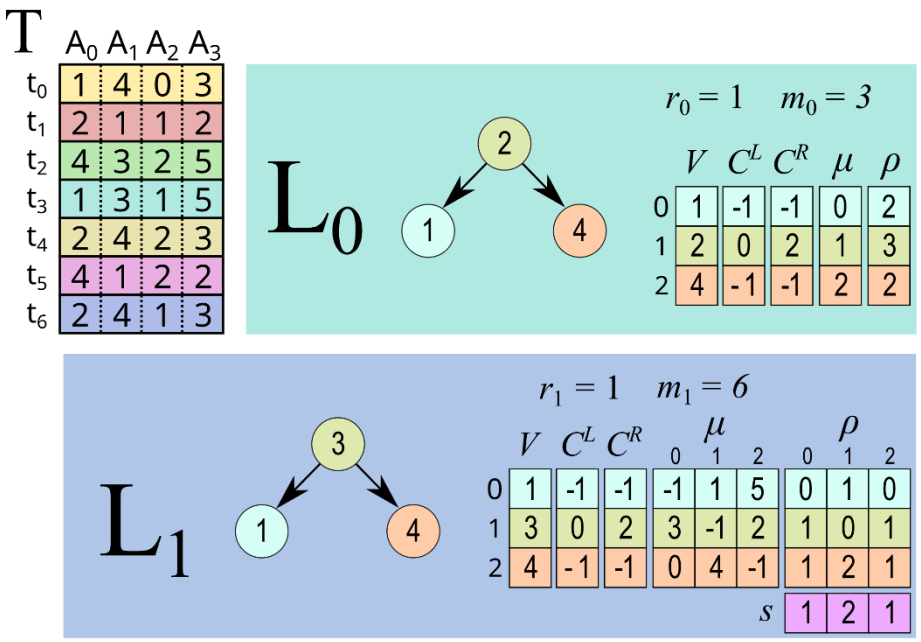

Figure 5: The previous example with additional structures to measure the support for AFDs.

LAYER BUILDING SEQUENCE

Since only one layer building step is enough to indicate the presence of a given functional dependency, the investigation of all viable attribute combinations can be done by setting up a sequence, where one layer is built in each step either on the one built in the previous step, or the one before that (except for the very first layers). An example can be seen for the so-called set containment lattice for 4 attributes (that start with A_0),

This sequence can be easily created by setting up a directed graph, where each node corresponds to an attribute number, and has a directed edge towards all nodes with higher attribute numbers. After that, a depth-first search is done on the graph: each time the processing enters a node, its attribute number is noted as the attribute of the layer that should be built in that step, and each time the processing has to step back (as there are no nodes with higher IDs to proceed to), then the sequence will step back to a previous attribute too. Figure 6 shows an illustration for the layer building sequence (LBS) on an N=4 attribute dataset. The red numbers denote which attribute is to be built in a given step, and the green ones are their base (i.e., the layer they are to be built upon). Notice that a) this can be simply generated by using the directed graph approach described above for A_0, then calculate the rest from it (by taking incrementing its value and taking its modular division with N) and b) it excludes half of the possible attribute combinations (without any negative consequences, as the order of attributes in the determinant set is irrelevant, i.e., $A_0A_1 \rightarrow A_2$ has the same support as $A_1A_0 \rightarrow A_2$). Therefore, the number of steps in an LBS is $N \cdot 2^{N-1}$, instead of 2^N.

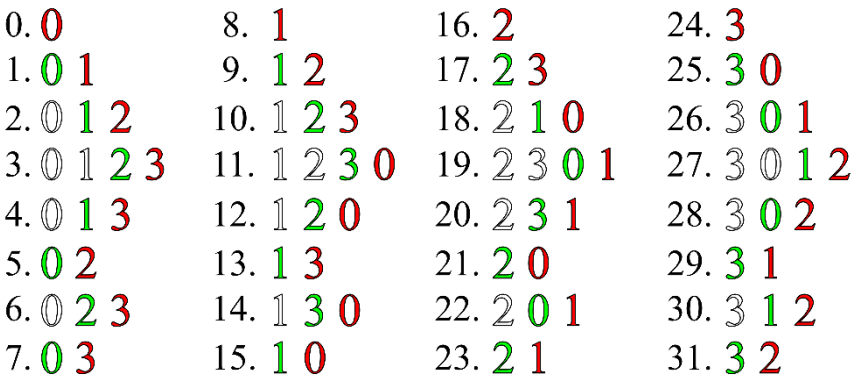

Figure 6: A layer building sequence for the lattice nodes (that start with 0). The red numbers denote a newly built layer (trained with the attribute values of that number), while the green ones are the base that the layer is built upon.

Normally, to be able to build a layer L_i upon a sequence of previously built layers, it is necessary to evaluate all tuples for each preceding layers in order to get the subset ID for each tuple in the last layer, unless the subset IDs are stored as well, in a 2D array H (with the same size as the dataset). This way, the layer building algorithm only has to regard what subset ID a given tuple got in the base layer (green in Figure 6) to build the new layer (and update the corresponding values in H).

THE OPERATION OF THE SIST METHOD

Considering the processes described previously, the proposed SIST works the following way:

1. Create the layer building sequence.
2. Set up the BST (V, C^L, C^R) for each attribute.
3. For each step in the layer building sequence, build a layer (μ, ρ, s) and maintain H, store each newly found FDs and AFDs (that has a support value above a given threshold).
4. Go through the set of found A/FDs, and delete the non-minimal ones.

The last step is necessary because the newly found A/FDs are not necessarily minimal (a subset of their determinant set is also determining the dependent attribute, e.g. the method finds both $A_0A_1A_3 \rightarrow A_2$ and $A_0A_1 \rightarrow A_2$, of which the former is non-minimal and can be disregarded).

RESULTS

The performance of the proposed SIST method has been investigated on 7 benchmark datasets (Dua et al., 2024), using an average laptop PC (Lenovo Legion 7 16ACHg6, AMD Ryzen™ 9 5900HX CPU, 32GB RAM, Nvidia GeForce RTX 3080 16GB). The datasets are the iris (Unwin & Kleinman, 2021), Wisconsin breast cancer (Mangasarian & Wolberg, 1990), chess King-Rook vs. King (Bain & Hoff, 1994), solar flare (Solar, 1989), glass identification (German, 1987), abalone (Nash et al., 1994) and seismic bumps (Sikora & Wrobel, 2010) datasets.

Table 1. shows the number of attributes and tuples for each dataset, as well as the size of the layer building sequence, and the number of strict FDs found. As it can be seen, the method found all FDs that are present in the datasets. The number of AFDs for 5 different support range are also shown.

Table 1: The 7 benchmark datasets used for testing, and the resulting A/FDs.

Dataset		**Iris**	**WBC**	**Chess**	**Solar Flare**	**Glass ID**	**Abalone**	**Seismic**
#Attributes (N)		*5*	*10*	*7*	*13*	*11*	*9*	*16*
#Tuples (P)		*150*	*683*	*28056*	*323*	*214*	*4177*	*2584*
LBS size		*32*	*5120*	*448*	*53248*	*11264*	*2304*	*524288*
Strict FDs		*4*	*20*	*1*	*9*	*170*	*137*	*374*
AFDs	**99-99.9%**	*2*	*143*	*0*	*13*	*233*	*740*	*2503*
	95-98.9%	*7*	*503*	*0*	*323*	*497*	*965*	*7909*
	90-94.9%	*2*	*1304*	*0*	*53248*	*622*	*1042*	*3851*
	85-89.9%	*11*	*2068*	*0*	*78*	*728*	*1087*	*1603*
	80-84.9%	*3*	*2539*	*1*	*1002*	*789*	*1126*	*3070*

Further details can be seen in the Table 2. Each dataset has been evaluated 1000 times and the required time and structure sizes have been averaged. The table also contains the highest structure sizes, as well as the highest process memory usage. The FD detection is measured without, while the AFD detection is measured with the additional structures (used for calculating the support for each given attribute combination). The latter has an at least 30% increase in structure size due to that, which also include a slight time increase (as the structures are needed to be created and maintained).

Remark: the structure sizes have been measured with a built-in function (*GetTotalMemory()*) of the programming language, while the process memory usage was taken using the Diagnostic Tools of MS. Visual Studio.

Table 2: The required time and memory of the FD and AFD analysis on the 7 benchmark datasets.

Dataset		**Iris**	**WBC**	**Chess**	**Solar Flare**	**Glass ID**	**Abalone**	**Seismic**
Required time [s]	***FD***	*0.003*	*0.246*	*0.6*	*0.75*	*1.85*	*47.072*	*2426.1*
	AFD	*0.003*	*0.326*	*0.719*	*0.989*	*3.056*	*70.808*	*3579.73*
Avg. structure size [MB]	***FD***	*0.698*	*0.45*	*4.44*	*0.756*	*3.56*	*67.54*	*18.4*
	AFD	*0.58*	*0.61*	*13.2*	*1.21*	*5.88*	*165.96*	*27.5*
Highest structure size [MB]	***FD***	*0.699*	*0.569*	*5.63*	*0.885*	*3.56*	*318.82*	*22.64*
	AFD	*0.58*	*0.62*	*35.26*	*1.21*	*9.278*	*318.33*	*42.92*
Highest process memory usage [MB]	***FD***	*9*	*12*	*29*	*11*	*19*	*629*	*391*
	AFD	*10*	*14*	*51*	*12*	*27*	*943*	*756*

The tests shows that the biggest factor on the required time is the LBS size, which is directly influenced by the number of attributes (in an exponential order). The size of the

structures on the other hand is more affected by the number of different values that the attributes can take.

The dependence on the number of attributes have also been investigated: different number of subsets of its attributes have been used ($N \in [3, 11]$, $P = 214$) as inputs for the SIST method, and the operational time has been averaged. The resulting exponential extraction time can be followed in Figure 7. However, the process memory usage (Figure 8.) has been a linear function of the attributes.

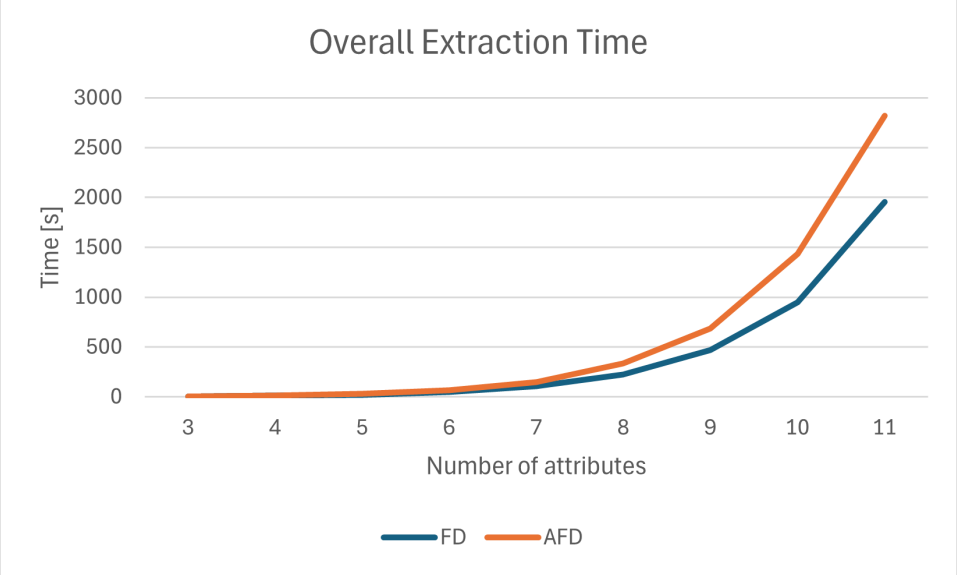

Figure 7: The overall FD and AFD extraction time for increasing attribute numbers.

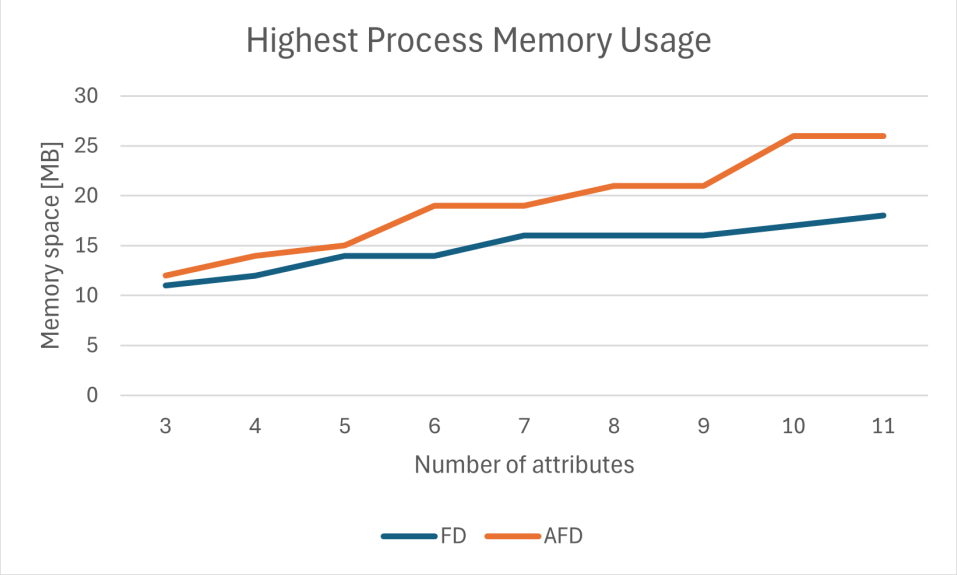

Figure 8: The highest process memory usage for FD and AFD extraction, for increasing attribute numbers.

The effect of increasing tuple numbers for the proposed system has also been investigated: random data has been generated from the glass identification dataset (taking the values from random rows for each attribute). The resulting average times can be seen in Figure 9, and the highest process memory usage in Figure 10. As it can be seen, they are both linear in the number of tuples.

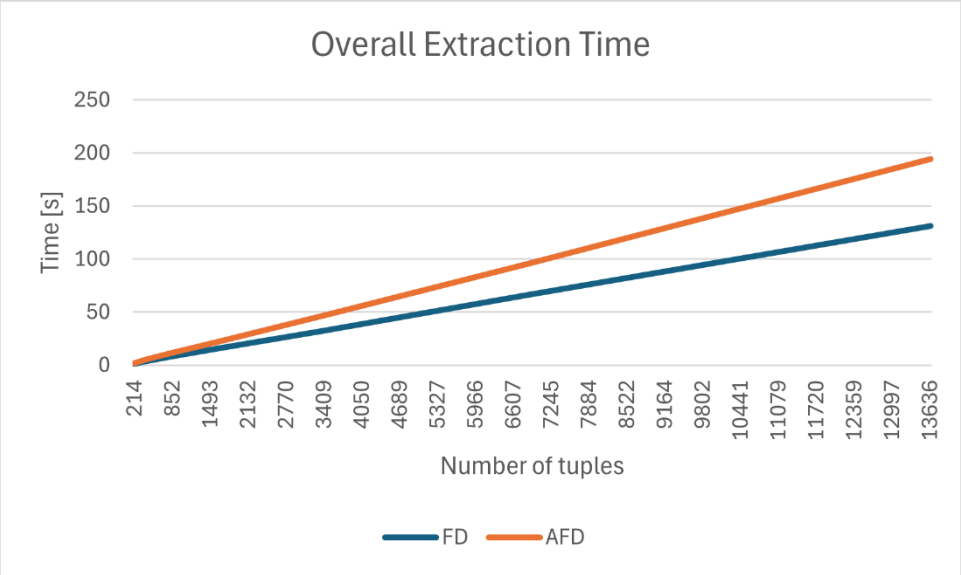

Figure 9: The overall FD and AFD extraction time for increasing numbers of tuples.

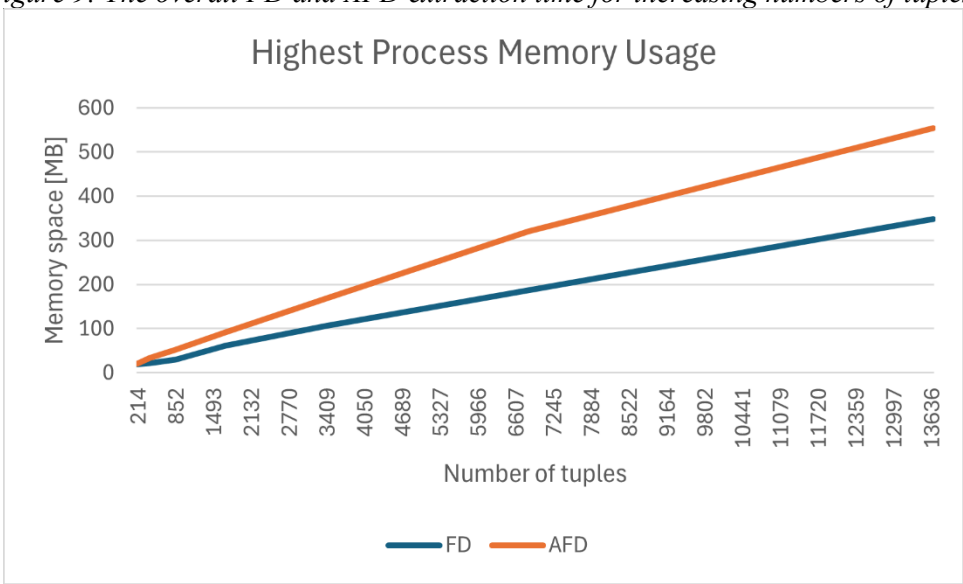

Figure10: The highest process memory usage for FD and AFD extraction, for increasing number of tuples.

COMPUTATIONAL AND SPATIAL COMPLEXITY

The computational complexity of the proposed SIST method is exponential in the number of attributes, as the tests have also indicated, due to the exponential number of layer building sequence steps that makes the algorithm investigate all significant attribute combinations. Each step involves the building of a single layer, using all P tuples and a BST with $\bar{V}$ nodes on average, thus, the operational time depends linearly on P and logarithmically on the average number of values among the attributes. Compared to TANE (Table 3.), it is much faster (as the latter depends on a quadratic order of P), but slower than FastFDs and SITs (both of which use heuristics to circumvent the exponential factor). Remark: $\bar{\varphi}$ is the average number of FDs, while D_{max} is the highest size of the mapped value domain among its attributes

(as SIT needs to map each value into an integer range, so distinguishing between values like 0.001 and 0.002 needs a scaling factor of 1000, potentially making the required memory too large).

In terms of spatial complexity, however, the proposed method is better than the other three, due to it only being dependent on *N* and *P* linearly. The predecessor SIT method is highly dependent on the domain size of the data, while the SIST is influenced by the typically much smaller $\bar{V}$.

Table 3: Comparison between the computational and spatial complexities of TANE, FastFD, SIT methods and the proposed SIST method.

Method	Computational Complexity	Spatial Complexity
TANE	$O\left(2^N \cdot (N \cdot P^{2.5})\right)$	$O\left(\frac{(N+P) \cdot 2^N)}{\sqrt{N}}\right)$
FastFDs	$O(P \cdot N^2 + P \cdot N^2 \cdot \log(P \cdot N^2))$	$O\left(N \cdot \frac{P \cdot (P-1)}{2}\right)$
SIT	$O(N^3 \cdot P \cdot \bar{\varphi})$	$O(N \cdot P \cdot D_{max})$
SIST	$O(N \cdot 2^{N-1} \cdot (P \cdot \log_2 \bar{V}))$	$O(N \cdot P \cdot \bar{V})$

CONCLUSION

In this paper, a new strict and approximate functional dependency method is presented that uses semi-balanced binary search trees combined with indexing tables to achieve a method that can indicate the presence of dependencies in its trained structure, instead of relying on pairwise comparisons between the tuples.

The goal of this preliminary research is to investigate the feasibility of the proposed method, as well as investigate its computational and spatial complexities. The tests conducted on multiple benchmark datasets have demonstrated that the proposed method is capable of finding all strict functional dependencies in a dataset and indicate approximate ones within given support ranges, and although its runtime is an exponential factor of the number of attributes (as no significant heuristics have been used yet), but its memory usage is a linear function of the number of tuples and attributes, so with suitable heuristics, it could be further developed into a method that can be advantageously used to analyse Big Data datasets.

In the next step of our research, the structure will be optimized to further reduce its spatial complexity, then we will investigate various heuristics to reduce the number of examined attribute combinations. For example, ignoring non-minimal A/FDs can potentially reduce the time requirement, for which a different LBS strategy is necessary. After that, the method will be implemented in Python so its performance can be compared to more modern methods (such as Pyro).

ACKNOWLEDGMENT

This publication is 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

Bain, M. & Hoff, A. (1994). Chess (King-Rook vs. King) [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C57W2S.

Dua, D., Graff, C., (2024). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml (accessed Mar. 1, 2024).

German, B. (1987). Glass Identification. UCI Machine Learning Repository. https://doi.org/10.24432/C5WW2P

Huhtala, Y., Kärkkäinen, J., Porkka, P., & Toivonen, H. (1999). TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, *42*(2), 100-111.

Kruse, S., & Naumann, F. (2018). Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment*, *11*(7), 759-772.

Liu, J., Li, J., Liu, C., & Chen, Y. (2010). Discover dependencies from data—a review. *IEEE Transactions on Knowledge and Data Engineering*, *24*(2), 251-264.

Mangasarian, O. L., & Wolberg, W. H. (1990). *Cancer diagnosis via linear programming*. University of Wisconsin-Madison Department of Computer Sciences.

Nash, W., Sellers, T., Talbot, S., Cawthorn, A., & Ford, W. (1994). Abalone [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C55C7W .

Sikora, M. & Wrobel, L. (2010). seismic-bumps [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5W902 .

Solar Flare [Dataset]. (1989). UCI Machine Learning Repository. https://doi.org/10.24432/C5530G .

Tusor, B., J.T. Tóth and A. R. Várkonyi-Kóczy, A.R. (2019). Functional Dependency Detection with Sequential Indexing Tables. In *2019 IEEE 23rd International Conference on Intelligent Engineering Systems (INES)*, *Gödöllő*, *Hungary*. 000307-000312. https://doi.org/10.1109/INES46365.2019.9109488.

Tusor, B., Tóth, J.T., and Várkonyi-Kóczy, A.R. (2019). SIT-based Functional Dependency Extraction. *Acta Polytechnica Hungarica*, *16*(*10*): 65-81.

Tusor, B., Tóth, J.T., and Várkonyi-Kóczy, A.R. (2019). Approximate Functional Dependency Mining with Sequential Indexing Tables. *In 2019 IEEE 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics (CINTI-MACRo)*, *Szeged, Hungary*, 000119-000124. https://doi.org/10.1109/CINTI-MACRo49179.2019.9105179

Unwin, A., & Kleinman, K. (2021). The iris data set: In search of the source of virginica. *Significance*, *18*(6), 26-29.

Várkonyi-Kóczy, A. R., Tusor, B., & Tóth, J. T. (2016, October). Active problem workspace reduction with a fast fuzzy classifier for real-time applications. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 004423-004428). IEEE.

Wyss, C., Giannella, C., & Robertson, E. (2001). Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *Data Warehousing and Knowledge Discovery: Third International Conference, DaWaK 2001 Munich, Germany, September 5–7, 2001 Proceedings 3* (pp. 101-110). Springer Berlin Heidelberg.