
Partial Inclusion Dependency Discovery

Partial Inclusion Dependency Discovery

Jakob Leander Müller

Universitätsmasterarbeit
zur Erlangung des akademischen Grades

Master of Science
(M. Sc.)

im Studiengang
Data Science

eingereicht am 20. Januar 2024 am
Fachgebiet Big Data Analytics der
Fakultät Mathematik und Informatik
der Philipps-Universität Marburg

Gutachter

Prof. Dr. Thorsten Papen-
brock

Betreuer

The amount of data that is being generated is growing constantly and at an ever increasing pace. All digital data is estimated to double about every two years [GR12]. This thesis will be focused around structured data, a subset of all digital data. Structured data refers to a type of data that is organized and formatted in a consistent manner, allowing for efficient search, retrieval, and analysis. More precisely, we will examine relational data, which is a type of structured data that is organized into tables or relations, where each table represents a set of entities or objects, and each row or tuple in the table represents a single instance of those entities. Example of efforts to collect relational data from the internet are the *Web Table Corpora*¹ or *Wikitables*². These project collect tables but do provide insights that could be extracted from the data. There is also publicly available data from governments, other researchers and private businesses.

One of the most fundamental concepts in relational data are foreign key relations[CFP82]. Foreign keys are a crucial aspect of relational databases as they help define the relationships between tables, maintain referential integrity, prevent errors, and improve the performance of operations pulling data from linked tables. They ensure that each record in one table corresponds to a valid record in another table, thereby promoting consistency and accuracy in the database. While foreign keys are not mandatory, they play a vital role in establishing clear relationships between tables and validating data as rows are added, updated, or removed. By linking data between tables, new insights can be extracted and previously hidden knowledge might get reviled. In today's economy, data profiling and therefore also the discovery of foreign key (and further inclusion dependencies), is a necessity which, if done by human experts, is connected to huge cost [HRO06].

In the ever-expanding field of data management and analytics, the accurate representation and comprehension of relationships within datasets stand as central challenge. Inclusion dependencies encapsulate hierarchical connections between attributes, playing a crucial role in the integrity and normalization of data [CFP82]. Understanding and discovering these dependencies have far-reaching implications

¹ <https://webdatacommons.org/webtables/>

² <http://websail-fe.cs.northwestern.edu/TabEL/>

for various applications, including database design [LV00], query optimization [Gry98], and data quality assurance. As the volume and complexity of data continues to increase, there is an ever growing need for advanced methodologies and tools that can extract inclusion dependencies inherent in datasets.

This master thesis embarks on a comprehensive exploration of inclusion dependency discovery. Inclusion dependencies capture the relationships between attributes by specifying that the values in one set of attributes must be included in another. While traditional database design principles rely on the normalization process to ensure the minimization of redundancy and enhance data integrity, the discovery and exploitation of inclusion dependencies provide a nuanced perspective on data relationships, offering insights that extend beyond conventional normalization techniques. We will not only discuss state of the art algorithms but further expand the search to partial inclusion dependencies. This special kind of inclusion dependencies allows for (small) errors and opens the door for finding inclusions dependencies in non-perfect datasets with human errors, spelling differences or historically grown deviations.

Insights generated through different partial thresholds are not merely academic, they have practical implications for companies and governments alike. If a partial inclusion dependency at a 99% threshold is found, organizations could use this information to check for impurity in the given attributes.

To ensure that the content of this thesis is readable by both experts and a general audience we need to formulate notations and definitions. These will reappear multiple times within the thesis and they are needed to formulate precise observations and draw conclusions. This sections sticks to the notation introduced by De Marchi et al. [MLP09].

A relational instance r of a relational schemata R carries tuples of values, typically donated as u or v . Using an attribute list taken from R , typically denoted as X or Y , we can perform a projection on R , thereby selecting a subset of attributes. We notate it by $R[X]$. The same is possible for tuples u . Writing $u[X]$ references the selection of values in the tuple.

► **Example 2.1 (Schemata and Attributes).** You can think of schema as a table and an attribute as a column in that table. If you think about picking multiple columns of some table, that would be an attribute list (X). Now every row has more than one value (a tuple of values, u), but every row has the exact same number of values (all tuple cardinalities are equal). ◀

Inclusion Dependencies (INDs) represent a fundamental concept, denoting formal relationships between attributes in a database schema. An IND specifies that the values within one set of attributes are inherently included within the values of another set of attributes.

► **Definition 2.2 (Inclusion Dependencies).** Given two relational instances r_i from R_i and r_j from R_j . An IND is defined as $R_i[X] \subseteq R_j[Y] \iff \forall u \in r_i, \exists v \in r_j \text{ such that } u[X] = v[Y]$. This condition can only hold if the cardinality of X is equal to the cardinality of Y . We further call the left hand side (here $R_i[X]$) the dependent attribute(s) and the right hand side the referenced attribute(s). ◀

► **Example 2.3 (Inclusion Dependencies).** The Tables 2.1 show two small examples of relational data. The left table (r_1) has the attributes *ID*, *Name*, *State* and *Age*. The right table (r_2) has the attributes *Name*, *State* and *Color*. We find the INDs $r_1[\text{Name}] \subseteq r_2[\text{Name}]$ and $r_2[\text{State}] \subseteq r_2[\text{State}]$. If r_1 had more entries and the

ID	Name	State	Age		Name	State	Color
1	Robin	HE	22				
2	Hannah	BW	24		Jakob	BW	Green
3	Christian	HE	36		Mareike	BE	Purple
4	Jakob	BW	24		Christian	HE	Red
5	Luka	BE	23		Leon	SH	Orange
6	Mareike	BE	22				
7	Leon	BE	27				

Table 2.1: Two example relational instances r_1 (left) and r_2 (right).

ID would just keep counting up, $r_1[Age] \subseteq r_1[ID]$ would become valid eventually. There are no INDs where the attribute cardinality is greater than one. ◀

The complexity of discovering inclusion dependencies forms one of the hardest problems in computer science. More precisely, the discovery of all inclusion dependencies is $W[3]$ -hard [BFS17]. This makes IND discovery one of the hardest problems in computer science. The number of possible candidates for each attribute size can be calculated. Notice that the formula assumes that all INDs from the layers before were valid. In natural language we search for the number of attribute combinations where each attribute is present at most once, allowing all permutations.

► **Definition 2.4 (Candidate Space).** Let α be the fixed integer size of all possible X_i . Let m be the number of attributes. Let k be the number of possible candidates ($X_i \subseteq X_j$ where $i \neq j$ given α).

$$k = \underbrace{\binom{m}{\alpha} \cdot \alpha!}_{\text{Left hand side combinations}} \cdot \underbrace{\binom{m-\alpha}{\alpha} \cdot \alpha!}_{\text{Right hand side combinations}}.$$

This formula holds if $\alpha \leq \lfloor \frac{n}{2} \rfloor$ else k is 0. ◀

In a multi schema setting this calculation becomes more difficult. We now need to consider which schema can form which inter schema candidates while allowing intra schema candidates.

► **Definition 2.5 (Candidate Space (Multi-Schema)).** We will first define a function $q_r(\alpha, \tau)$, which computes the candidate a single schema (r) can produce, given α , the size of combinations and τ an indicator, whether or not the schema was already used for the other side.

$$q_r(\alpha, \tau) = \begin{cases} 0, & \text{if } \alpha > |r| \vee (\alpha > \lfloor \frac{|r|}{2} \rfloor \wedge \tau) \\ \binom{|r|}{\alpha} \cdot \alpha!, & \text{if not } \tau \\ \binom{|r|-\alpha}{\alpha} \cdot \alpha!, & \text{if } \tau \end{cases}$$

The calculate the possible candidates k for a fixed α over schema r_1, \dots, r_n we need to compute:

$$k = \sum_{i=1}^n q_{r_i}(\alpha, FALSE) \cdot \sum_{i=1}^n q_{r_i}(\alpha, TRUE)$$



► **Definition 2.6 (Partial Inclusion Dependencies).** A partial inclusion dependency (pIND) is written as $r_1[X] \subseteq_{\rho} r_2[Y]$ where $\rho \in (0, 1]$ and the reaming notation is analog to 2.2. Here, the ρ interval is not including 0 since this would mean everything is a pIND of everything else, which is a trivial case. Further this notation refers to lists of tuples and takes the cardinality of duplicates into consideration. For the pIND $r_1[X] \subseteq_{\rho} r_2[Y]$ to be valid

$$\frac{|r_1[X] \cap r_2[Y]|}{|r_1[X]|} \geq \rho$$

needs to be true.



This definition uses the \cap operator, which is usually used in set theory and defined for sets. Here the operator conducts the operation *remove all elements from the dependent attribute, that are not present in the referenced attribute*. Therefore the referenced attribute could be a set in a sense that the duplication distribution only matters for the dependent attribute.

In the proposed algorithms there is the option of considering duplicate cardinalities. If not explicitly mentioned otherwise this thesis always refers to partial inclusions that consider duplicate cardinality.

► **Theorem 2.7 (Partial Inclusion Dependency Properties).** Like inclusion dependencies, partial inclusion dependencies also full fill the reflexive rule. For any $\rho \in (0, 1]$ the partial inclusion dependency $r_i[X_j] \subseteq_{\rho} r_i[X_j]$ is valid.

$$\frac{|r_i[X_j] \cap r_i[X_j]|}{|r_i[X_j]|} \geq \rho$$

$$\frac{|r_i[X_j]|}{|r_i[X_j]|} \geq \rho$$

$$1 \geq \rho$$

Since ρ is upper bounded by 1 the last statement will always be true.

Contrary to INDs, pINDs do not generally respect transitivity if $\rho < 1$. We will proof this claim by contradiction. Assume $r_1[X] = [1, 2, \dots, 100]$, $r_2[Y] = [2, \dots, 1000]$, $r_3[Z] = [10, 11, \dots, 1000]$. If transitivity would hold for any ρ , then we should find that for $\rho \in (0, 1]$ where $r_1[X] \subseteq_\rho r_2[Y]$, $r_2[Y] \subseteq_\rho r_3[Z]$ are valid, $r_1[X] \subseteq_\rho r_3[Z]$ also needs to be valid. For the given example, if $\rho = 0.95$, we find a contradiction.

Lastly, INDs and also pINDs respect projection. We will now outline a proof for this claim. Consider the attributes $r_1[X], r_2[Y], r_3[Z], r_4[W]$ where $r_1[X]$ and $r_2[Y]$ are in the same relation and $r_3[Z]$ and $r_4[W]$ are in the same relation. Assume $r_1[X], r_2[Y] \subseteq_\rho r_3[Z], r_4[W]$ is valid for some $\rho \in (0, 1]$. If projection holds, this implies that $r_1[X] \subseteq_\rho r_3[Z]$ and $r_2[Y] \subseteq_\rho r_4[W]$ have to be valid as well. If we now only consider the portion (with reduced size $\rho\%$) which satisfies $r_1[X], r_2[Y] \subseteq_1 r_3[Z], r_4[W]$ we can use the known properties for INDs and conclude that for at least $\rho\%$ $r_1[X] \subseteq_1 r_3[Z]$ and $r_2[Y] \subseteq_1 r_4[W]$ has to be valid. This also directly implies that $r_1[X] \subseteq_\rho r_3[Z]$ and $r_2[Y] \subseteq_\rho r_4[W]$ will be true if the remaining $(1 - \rho)\%$ values are added again. This property is very important for search space pruning, which is the single most important task for (p)IND discovery [Liu+10]. ◀


► **Example 2.8 (Partial Inclusion Dependencies).** Let us consider the two attributes $r_1[X] = [1, 2, 3, 4]$ and $r_2[Y] = [1, 1, 2, 3]$. First we can conclude, that $r_2[Y] \subseteq r_1[X]$ is an inclusion dependency, since all values present in $r_2[Y]$ also occur in $r_1[X]$. This also directly causes $r_2[Y] \subseteq_{1.0} r_1[X]$ to be a valid pIND. We will now calculate the maximal partial thresholds. Like inclusion dependencies, partial inclusion dependencies are not symmetrical, this requires us to perform two calculations. We already discovered $r_2[Y] \subseteq_{1.0} r_1[X]$, which implies the maximal threshold is 1. Let us use the proposed formula for the other direction.

$$\frac{|r_1[X] \cap r_2[Y]|}{|r_1[X]|} \geq \rho$$

$$\frac{|[1, 2, 3, 4] \cap [1, 1, 2, 3]|}{|[1, 2, 3, 4]|} \geq \rho$$

$$\frac{|[1, 2, 3]|}{|[1, 2, 3, 4]|} \geq \rho$$

$$\frac{3}{4} \geq \rho.$$

We have found that when the threshold ρ is less or equal to 0.75, $r_1[X] \subseteq_{\rho} r_2[Y]$ is valid. 

The complexity of discovering partial inclusion dependencies is inherit from the base problem. Since the complexity is calculated under a worst case assumption, it does not change when switching into the partial setting. The thesis we will discuss in detail how the time complexity behaves under varying thresholds.

2.1 Technologies

Discovering inclusion dependencies is not a new problem. Over the years researchers have used a variety of technologies to improve performance. Starting with storage the two most common assumptions are either a relational database or structured storage files (e.g. CSV or TSV files). There are algorithms which rely on SQL for IND candidate validation [BB95]. Still, the techniques used in the algorithms are not dependent on the input format or language of implementation. This is why we discuss algorithms without setting constraints on the input format or validation strategy. The thesis will assume the input to always be static files in CSV format. Nevertheless there have been efforts to collect all kinds of algorithms and integrate them into a single framework [Dür+19]. The Metanome platform [Pap+15a] provides the framework for the stated implementations and the thesis will use the platform to execute tests on existing algorithms as well as contributing new algorithms and an implementations to run and display results of partial inclusion dependency algorithms. Metanome and existing implementation are written in Java which lead me to also pick Java as my language of choice.

2.2 Related Work

Inclusion dependencies (INDs) are a highly influential concept in both database research and practice, with a wide range of contributions and applications. The introduction already provided some insight into their diverse application areas of INDs. In this section, we will focus on the key achievements related to the

implication problem of INDs. We will go over different algorithms and discuss their unique features.

In 1981 INDs started as a general notation of referential integrity, which was already a well established concept back then [Dat81]. Casanova et al. presented a paper on the inference rules of INDs [CFP82]. Three axioms were introduced: *reflexivity*, *transitivity* and *projection and permutation*. The application of these rules to partial INDs will later be discussed in detail (Section 2.7). The paper further proofed that the discovery of INDs is PSPACE-complete if there is no limit on the size of inclusions. Publications typically fall into three groups of algorithms, foreign key discovery algorithms, unary IND discovery and n-ary IND discovery [Pap17].

In 1995 Bell and Brockhausen [BB95] propose a graph-based approach to represent the relationships between attributes, allowing for a more efficient exploration of the search space. The algorithm is initiated with a directed graph, wherein all possible edges, which could not be pruned by statistical measures, are included. A directed edge in the graph represents an inclusion dependency, which is read as the edge from A_i to A_j ($A_i \rightarrow A_j$) represents the IND $A_i \subseteq A_j$. It then proceeds to remove those edges that failed the IND check. To determine the validity of an edge, the algorithm checks for transitivity, which enables it to answer whether a dependency could exist between two variables based on their relationships with a third variable, which was tested previously. If it is ascertained that a dependency is impossible, the algorithm skips the test and directly removes the given edge, thereby reducing the overall computational cost. The approach presented by Bell and Brockhausen for unary IND discovery has both reusable aspects and downsides. The algorithm's candidate generation technique, which uses data statistics such as data types and min-max values, can be reused in other discovery algorithms. This preprocessing step reduces the number of candidates that need to be validated and further reduce the storage overhead needed to store candidates. However, the validation technique used in the algorithm, which relies on SQL join-statements and requires accessing the data on disk, is infeasible for larger candidate sets. This limits the scalability of the approach and makes it less practical for large-scale data sets. Additionally, the need to store the data in a database and access it on disk for validation can add to the computational cost and time required for the discovery process.

The *SPIDER* algorithm is a disk-backed, all-column sort-merge join with early termination used for the discovery of inclusion dependencies [Bau+06]. It sorts the values of each attribute, removes duplicate values, and writes the results to disk in

the first phase. In the second phase, it performs the actual inclusion dependency discovery by using a pointer for each file and validating all candidates at the same time. A major advantage is, that in this setting every value only needs to be read a single time from disk, which greatly reduces the I/O bottleneck. The Spider algorithm has been the subject of experimental evaluation and is considered one of the efficient techniques for unary IND discovery. Still, there are drawback if the data set is too big to be sorted in main memory or if the number of simultaneously open files allowed by the OS system is reached [Pap+15b].

In 2009 Bauckmann et al. also proposed a partialized version of *SPIDER* in a section of the same paper. The authors found that there where surprisingly many partial inclusions (under a 5% threshold) in their test data sets. To find partial inclusion dependencies, they first count how many distinct violations are present and in a second step consider the amount of duplicates for not included values. This means their algorithm does not immediately stop once a single validation has been found but only after an added counter surpasses a given threshold. The paper is not particular clear on how the number of duplicates is stored/retrieved and additionally does not analyse the computational effect of these changes and with the original source code being lost, this a approach can only be verified using a best guess approach.

The *SAWFISH* algorithm [KPN23], published in 2023, is designed for identifying similarity inclusion dependencies (sINDs) within datasets, introducing a novel perspective on inclusion dependencies (INDs). While traditional INDs assume error-free data, *SAWFISH* incorporates a similarity measure to accommodate minor errors like typos. Given a similarity threshold ω a sIND is valid if and only if for all tuples in the left hand side there exists a tuple in the right hand side which has at least a similarity of ω under a set similarity measure. The authors used the edit distance as well as the normalized edit distance. Through preprocessing, metadata generation, and a sliding-window approach, *SAWFISH* successfully identifies and validates sIND candidates using an inverted index, providing a valuable tool for database applications despite dirty data challenges.

To understand the performance of the proposed algorithms it is crucial to perform testing on a variety of data sets. For this purpose we will gather some real word data sets. Further we will create synthetic data sets that aim on edge cases to see if the performance is strongly dependent on structural assumptions.

3.1 Real World Data Sets

There are many sources for csv or tsv files online. I have decided to gather data from the US Government³, the European Union⁴, Kaggle⁵, Musicbrainz⁶, and Eurostat⁷. Further data set sources may be added. Related research papers sometimes discuss the origin of the used data, do not discuss the structure of the data they use [Bau+06; Dür+19; Pap17; Ros+09]. In order to understand the resulting algorithm performance, we believe it is crucial to examine the data which is tested against. In this section we will discuss the data used and later try to understand why an algorithms performance may vary over different test sets.

3.2 Synthetic Data Sets

To evaluate the proposed algorithms under detailed aspects, we will generate synthetic data sets. The strategies and claims are based on [Jor+22] synthetic data can be defined as *data that has been generated using a purpose-built mathematical model or algorithm, with the aim of solving a (set of) data science task(s)*. While we will not try to train a model with the synthetic data, it is still of great use for us, since we have absolute knowledge about the underlying structures. The decision is based on the fact that there is a lot of real word data available, since open data is a growing market which expected to grow even further [EUO22]. Synthetic data on the other hand enables us to evaluate the algorithm performances on edge cases, which we

³ data.gov

⁴ data.europa.eu

⁵ [Kaggle.com](https://kaggle.com)

⁶ musicbrainz.org

⁷ ec.europa.eu

may not be able to find in the selection of real world data sets.

To test certain edge cases of the proposed algorithms, we will construct various edge case data sets. The *SameSame* dataset consists of 32 attributes and 250.000 records. Each attribute carries the numbers 1 to 250.000 in the natural order. This means every attribute is a (partial) inclusion dependency of every other attribute. The same obviously also holds for combinations of columns. We will now calculate the expected number of (p)INDs in each layer. Since all candidates are perfect matches, the chosen threshold ρ will not influence the number of pINDs. Table shows the number of candidates/pINDs for the *SameSame* dataset. The edge case to test here is, how well the algorithm can understand equality relations and prune the candidate space. While this may seem like an unlikely edge case we will also investigate how often this happens in real world data sets.

Another source of synthetic data will be the TPC (Transaction Processing Performance Council) Benchmarks ⁸. The TPC Benchmarks are a set of standardized and vendor-neutral performance benchmarks used to evaluate the processing and database capabilities of different systems. These benchmarks are designed to model various types of workloads. The TPC-E benchmark, for example, models a brokerage firm with customers who generate transactions related to trades, account inquiries, and market research, while the TPC-C benchmark is intended to model a medium complexity online transaction processing workload, patterned after an order-entry system with skewed access within individual data types/relations. Using scaling factors, a user can define the size of the synthetic database themselves. This enables us to examine the algorithm performances in a very controllable setting.

8 tpc.org

In conclusion, this research will aim to answer several key questions and address specific tasks in order to provide a comprehensive understanding of partial inclusion dependencies (pINDs) in real-world data. These research questions and tasks may evolve throughout the course of the study.

Since pINDs are a mostly unexplored topic, the central research question revolves around finding an efficient pIND discovery algorithm. Modifying the BINDER algorithm to enable partial IND discovery, could be a first solution and provide a benchmark for further implementations. Within BINDER we need to adjust the validator to respect partial INDs. Since the source code for the original BINDER algorithm is not well structured, we need to perform a rewriting of possible most of the code.

As discussed in Section 2.2, the authors of *SPIDER* also proposed a partial version of their algorithm. We will implement and publish their proposed algorithm.

Next to *BINDER* and *SPIDER* we will spent the majority of the research on finding an efficient method for pIND discovery. A first idea is, to use a graph structure, that carries meta-data, to invalidate candidates and keep track of existing relations. The thesis will also compare the time it takes to discover pINDs to the time it takes to discover traditional inclusion dependencies (INDs). This will involve investigating the impact of the threshold value on the execution time of pIND discovery algorithms and determining which strategies proposed in past literature can be applied in a partial setting and which strategies will not work.

We will seek to determine the prevalence of pINDs in real-world data and investigate the potential benefits of discovering them. This will involve identifying the datasets in which pINDs are commonly found, as well as establishing a threshold for pIND discovery that is applicable across different dataset domains. We do not expect to find statistically significant findings, since the testing size will most likely be too small for that. The investigation rather aims on getting a first glimpse on the value pINDs could provide.

Bibliography

- [Bau+06] Jana Bauckmann, Ulf Leser, Felix Naumann, and Véronique Tietz. **Efficiently detecting inclusion dependencies**. In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE. 2006, 1448–1450 (see pages 8, 11).
- [BB95] Siegfried Bell and Peter Brockhausen. **Discovery of data dependencies in relational databases**. Citeseer, 1995 (see pages 7, 8).
- [BFS17] Thomas Bläsius, Tobias Friedrich, and Martin Schirneck. **The parameterized complexity of dependency detection in relational databases**. In: *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017 (see page 4).
- [CFP82] Marco A Casanova, Ronald Fagin, and Christos H Papadimitriou. **Inclusion dependencies and their interaction with functional dependencies**. In: *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*. 1982, 171–176 (see pages 1, 8).
- [Dat81] CJ Date. **Referential integrity**. In: *Proceedings of the seventh international conference on Very Large Data Bases-Volume 7*. 1981, 2–12 (see page 8).
- [Dür+19] Falco Dürsch, Axel Stebner, Fabian Windheuser, Maxi Fischer, Tim Friedrich, Nils Strelow, Tobias Bleifuß, Hazar Harmouch, Lan Jiang, Thorsten Papenbrock, and Felix Naumann. **Inclusion Dependency Discovery: An Experimental Evaluation of Thirteen Algorithms**. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. ACM, 2019, 219–228. DOI: [10.1145/3357384.3357916](https://doi.org/10.1145/3357384.3357916) (see pages 7, 11).
- [EUO22] EUOpenData. *Open Data Impact - European Union*. <https://data.europa.eu/en/publications/open-data-impact>. Accessed: 2023-12-08. 2022 (see page 11).
- [GR12] John Gantz and David Reinsel. **The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east**. *IDC iView: IDC Analyze the future 2007:2012* (2012), 1–16 (see page 1).
- [Gry98] Jarek Gryz. **Query folding with inclusion dependencies**. In: *Proceedings 14th International Conference on Data Engineering*. IEEE. 1998, 126–133 (see page 2).

- [HRO06] Alon Halevy, Anand Rajaraman, and Joann Ordille. **Data integration: The teenage years**. In: *Proceedings of the 32nd international conference on Very large data bases*. 2006, 9–16 (see page 1).
- [Jor+22] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. **Synthetic Data—what, why and how?** *arXiv preprint arXiv:2205.03257* (2022) (see page 11).
- [KPN23] Youri Kaminsky, Eduardo HM Pena, and Felix Naumann. **Discovering Similarity Inclusion Dependencies**. *Proceedings of the ACM on Management of Data* 1:1 (2023), 1–24 (see page 9).
- [Liu+10] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. **Discover dependencies from data—a review**. *IEEE Transactions on Knowledge and Data Engineering* 24:2 (2010), 251–264 (see page 6).
- [LV00] Mark Levene and Millist W Vincent. **Justification for inclusion dependency normal form**. *IEEE Transactions on Knowledge and Data Engineering* 12:2 (2000), 281–291 (see page 2).
- [MLP09] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. **Unary and n-ary inclusion dependency discovery in relational databases**. *Journal of Intelligent Information Systems* 32 (2009), 53–73 (see page 3).
- [Pap+15a] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. **Data Profiling with Metanome**. *Proc. VLDB Endow.* 8:12 (Aug. 2015), 1860–1863. ISSN: 2150-8097. DOI: [10.14778/2824032.2824086](https://doi.org/10.14778/2824032.2824086). URL: <http://dx.doi.org/10.14778/2824032.2824086> (see page 7).
- [Pap+15b] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. **Divide & conquer-based inclusion dependency discovery**. *Proceedings of the VLDB Endowment* 8:7 (2015), 774–785 (see page 9).
- [Pap17] Thorsten Papenbrock. **Data profiling-efficient discovery of dependencies**. PhD thesis. Universität Potsdam, 2017 (see pages 8, 11).
- [Ros+09] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. **A machine learning approach to foreign key discovery**. In: *WebDB*. 2009 (see page 11).