Assessing the Existence of a Function in a Dataset with the g3 Indicator

Pierre Faure-Giovagnoli, Jean-Marc Petit, Vasile-Marian Scuturici

Assessing the Existence of a Function in a Dataset with the g_3 Indicator

Pierre Faure--Giovagnoli[1,2]
[1]*Univ Lyon, INSA Lyon, CNRS, UCBL*
LIRIS UMR 5205, Villeurbanne, France
[2]*Compagnie Nationale du Rhône, Lyon, France*
pierre.faure--giovagnoli@liris.cnrs.fr

Jean-Marc Petit
Vasile-Marian Scuturici
Univ Lyon, INSA Lyon, CNRS, UCBL
LIRIS UMR 5205, Villeurbanne, France
firstname.lastname@liris.cnrs.fr

Abstract

Taking domain knowledge into account is a long-standing issue in AI, especially nowadays where huge amounts of data are collected in the hope of delivering new insights and value. Let us consider the following scenario. Let $D(y, x_1, \ldots, x_n)$ be a dataset, Alice a data scientist, Bob a domain expert and $y = f(x_1, \ldots, x_n)$ a function known by Bob from his background knowledge. We are interested in the following simple yet crucial questions for Alice: how to define the satisfaction of f in D and how difficult is it to measure that satisfaction? It turns out that those problems are related to functional dependencies (FDs) and especially FD measurements used to quantify their satisfaction in a dataset such as the g_3 indicator.

In this paper, we examine the computation of g_3 with crisp FDs (aka. exact FDs) and a large class of non-crisp FDs replacing strict equality by more flexible predicates. Interestingly, it is known that the computation of g_3 with crisp FDs is polynomial but turns out to be NP-Hard for non-crisp FDs. In this paper, we propose different exact and approximate solutions for the computation of g_3 for both types. First, for crisp FDs with very large datasets, we propose solutions based on uniform and stratified random sampling. Second, for non-crisp FDs we present a detailed computation pipeline with various computation optimizations, including approximation algorithms and adaptations of recent developments in sublinear algorithms for NP-Hard problems.

We also propose an in-depth experimental study of the algorithms presented in terms of time performances and approximation accuracy. All the algorithms are also made available through FASTG3, an open-source Python library designed to be intuitive and efficient thanks to an underlying C++ implementation.

Keywords

domain knowledge; function; functional dependency; crisp; non-crisp; coverage; g3; error; confidence; NP-hardness; computation;

I. Introduction

When collaborating with domain experts with the aim of confronting their knowledge with real data, data scientists need an efficient way to express and measure their theoretical model against a dataset. Inspired by a current collaboration with the industry, we consider the toy dataset (r_{toy}) in Table I presenting various values of a hydropower turbine. Notably, we consider the three following attributes: the incoming flow (measured in m^3s^{-1}), the elevation of the waterfall (m) and the power produced (MW). From collaboration with domain experts, let us assume that, at least in theory, the power is completely dependant on the flow and the elevation according to the following model expressed as a function:

$$\text{power} = f_{\eta,\rho}(\text{flow}, \text{elevation}) = \eta \cdot \rho \cdot \text{flow} \cdot \text{elevation} \tag{1}$$

with η the turbine efficiency specific to the machine and ρ the water density, 2 fixed parameters.

Therefore, to assess the proper functioning of their turbines and gain insights into potential technical optimizations, it is of interest to evaluate the veracity of function 1 against on-site recorded data in the form of r_{toy}.

TABLE I: RELATION r_{toy}

id	flow	elevation	power
t0	2.6	10.1	23.3
t1	2.5	10.2	22.9
t2	2.5	10.2	23.0
t3	2.6	10.0	23.4
t4	2.7	10.0	24.3
t5	2.7	10.1	24.5

Functions are deterministic by nature: *there can only be one output for a given input*. To express such a constraint in data, functional dependencies (FDs) have proven their effectiveness by offering a comprehensive framework to express constraints between sets of attributes. Introduced in [1], FDs have been extensively used to express restrictions and to verify integrity in

databases but their role has gradually extended to many tasks from data cleaning [5] to data mining [36] or query optimization [34]. More broadly, FDs and their many relaxations can be used to express and analyze most relationships in a dataset.

In its original definition, a crisp FD φ defined over a schema R is an expression in the form $X \rightarrow C$ with $X \cup \{C\} \subseteq R$ where X is called the antecedent and C the consequent. φ is said to be *satisfied* in a relation r of R when for all pairs of tuple (t_1, t_2) in r, if $t_1[A] = t_2[A]$ for all A in X, we have $t_1[C] = t_2[C]$. In other words, φ expresses formally that the values of the antecedents in r should completely define the values of the consequent. *In the toy dataset presented above, the function examined could be naturally expressed as follows:*

$$\varphi_{crisp} : \text{flow}, \text{elevation} \rightarrow \text{power}$$

However, this definition of satisfaction is known to be too strict to portray many real-life scenarios. Thus, it is often necessary to quantify the partial agreement of an FD in data rather than just assessing its perfect satisfaction for all tuples. To this end, a measure of satisfaction (a.k.a coverage) needs to be chosen for φ in r. Initially defined in [30], the most common indicator for this purpose is known as g_3. More precisely, $g_3(\varphi, r)$ corresponds to the smallest proportion of tuples to be removed from relation r for φ to hold in r. In the literature, the g_3 indicator is often named g_3-error [7] or just error [25] while its opposite $(1 - g_3)$ is often called confidence (abrv. conf) [11], [20], [43]. *Due to errors in sensor recordings or temporary perturbations in the turbine (eg. various elements such as branches often succeeding to passing upstream filters), it is very unlikely that an FD such as φ_{crisp} will be valid on the whole dataset despite being almost accurate. For instance, we may consider φ_{crisp} to be accurate enough if $g_3(\varphi_{crisp}, r_{toy})$ is below 5%.*

In addition, it is of interest to extend the use of g_3 to other types of FDs. In this paper, we focus on relaxations on the attribute's value comparison where the strict equality imposed by crisp FDs is replaced by predicates as proposed in [7], [9], [45]. This powerful relaxation makes it possible to capture a more refined notion of closeness directly linked to the problem at hand and the domain knowledge available. *In the case of r_{toy}, many FDs would turn out to be trivially satisfied since all X values are likely to be unique as they are continuous measures. It appears that a major piece of information has been forgotten in φ_{crisp}: sensor uncertainty. Indeed, all sensor measures have an associated uncertainty which makes strict equality insufficient to fully capture the subtlety of the data. Along with domain experts, it is possible to define a more accurate non-crisp FD such as:*

$$\varphi_{ncrisp} : [\text{flow} \pm 0.05 \cdot \text{flow}], [\text{elevation} \pm 0.05] \rightarrow [\text{power} \pm 0.01]$$

The only difference is the replacement of the equality by an appropriate predicate. With τ_a and τ_r being respectively the absolute (eg. elevation and power) and relative (eg. flow) uncertainties, for two values a and b of a given attribute, the crisp equality could be replaced by a predicate such as:

$$|a - b| \leq \tau_a + \tau_r \cdot max(|a|, |b|) \tag{2}$$

For r_{toy}, the pair (t_1, t_2) is the only one to violate φ_{crisp} despite some other pairs of tuples being very close (eg. (t_3, t_4)). However, integrating uncertainties by using φ_{ncrisp} displays the possible difficulties of real-life data in matching the target function: $\{(t_0, t_5), (t_1, t_2), (t_3, t_4)\}$. In this trivial case, it is sufficient to remove one tuple in each pair to get the smallest satisfying subset such that $g_3(\varphi_{crisp}, r_{toy}) = \frac{1}{6}$ and $g_3(\varphi_{ncrisp}, r_{toy}) = \frac{3}{6} = 0.5$. In this latter case, it means that half of the tuples need to be removed from r_{toy} for φ_{ncrisp} to be satisfied. Such a high error would require discussion with domain experts: what are the possible causes of such deviations from the model?

However, if g_3 can be computed in polynomial time in the case of crisp FDs such as φ_{crisp}, introducing relaxations on value comparison such as the one presented in φ_{ncrisp} shifts it into the NP-Hard problem class by reduction from the Minimum Vertex Cover (MVC) [43].

Moreover, g_3 is at the core of many FD mining algorithms [30], [49], [25], [8] and new uses have recently emerged in the literature. Notably, we consider the supervised learning case of a target C to be predicted from a set of features X using a learning algorithm on r. With strict equality predicates, [31] points out $\text{conf}(X \rightarrow C, r)$ to be an upper-bound on the accuracy of any model built on r. Equally, [17] presents ADESIT, a tool based on the g_3 indicator to help domain experts predict the success of the learning process.

In summary, we present the following contributions:

- **Crisp FDs.** It is known that the computation of g_3 with crisp FDs is easy for small datasets [30]. In this paper, we propose practical solutions for its computational scalability with larger datasets, a subject that has received only little attention in the literature. Thus, we compare the scalability of two exact algorithms optimizing memory and time complexity respectively for computing g_3 with crisp FDs. For very large datasets where exact algorithms find their limit, we analyze the theoretical guarantees of uniform random sampling and study advanced sampling schemes to allow for computation

scalability. Notably, we propose an improvement on a stratified sampling algorithm proposed in [11], an improvement which provides excellent results in practice.

- **Non-crisp FDs.** The computation of g_3 with differential dependencies (a special type of FDs) has been proven to be NP-Hard [43]. We generalize this result to a larger class covering many known FD extensions in the literature, and present, to the best of our knowledge, the first full pipeline for computing the g_3 indicator with relaxed equality. We examine its computational scalability by presenting connections to similar problems in the literature, especially for the violating pairs enumeration process which is, in practice, the bottleneck of the operation. We also propose solutions for its exact computation and, for very large datasets, we adapt approximation algorithms and recent developments in sublinear algorithms for NP-Hard problems [50], [37].
- **Experiments.** We conduct comprehensive experiments through a study of time performance and approximation accuracy. Notably, we demonstrate that the proposed algorithms and optimizations can be used for very large datasets with reasonable computation times while keeping a good level of accuracy for the expected results. We also detail what dataset specificities and problem parameters impact the computation time. All the algorithms are also made available through FASTG3, an open-source Python library for computing g_3 providing efficient and scalable C++ implementations [4] with an intuitive API.

Firstly, we present the g_3 indicator and its associated semantics. Then, we propose solutions for its computation in the case of crisp and non-crisp FDs with solutions for its exact and approximate computation. Finally, we experiment with multiple datasets to analyze the time processing performances and the accuracy of the approximate algorithms.

II. Preliminaries

We consider a relation r of size (number of rows) n defined over schema $R = \{A_1, ..., A_m, C\}$. We consider an FD φ in the form $X \rightarrow C$ with $X \cup \{C\} \subseteq R$. Firstly, we define the satisfaction of crisp FDs as proposed by Armstrong in [1] (also called *exact* or *classical* FDs). φ_{crisp} is an example of such an FD.

Definition II.1 (Satisfaction of crisp FDs)**.** *φ is satisfied in a relation r (noted $r \models \varphi$) if:*

$$\forall t_1, t_2 \in r, \forall A_i \in X, t_1[A_i] = t_2[A_i] \Rightarrow t_1[C] = t_2[C]$$

As stated before, it is well-known that the strict equality imposed by crisp FDs is often too narrow to capture intricate real-life phenomena and numerous extensions have been proposed [7]. Indeed, when discussing with domain experts, the data scientist often faces the situation where two very close values are considered equal from a practical point of view. *A small displacement in the cog of a turbine is not always sufficient to drive the next one, a small delta in measured voltage may be due to static electricity and two flow values may be equal in real life but different in their measurement due to sensor uncertainty...* We focus on a well-known class of FDs allowing to relax the equality by attribute-wise predicates able to capture this notion of closeness and call them non-crisp FDs.

Definition II.2 (Satisfaction of non-crisp FDs)**.** *We equip each attribute A_i in $X \cup \{C\}$ with a predicate ϕ_i such that:*

$$\phi_i : \text{dom}(A_i) \times \text{dom}(A_i) \rightarrow \{\text{true}, \text{false}\}$$

We assume ϕ_i to be computable in polynomial time in the size of its input. Thus, φ is satisfied in a relation r (noted $r \models_{nc} \varphi$) if:

$$\forall t_1, t_2 \in r, \bigwedge_{A_i \in X} \phi_i(t_1[A_i], t_2[A_i]) \Rightarrow \phi_c(t_1[C], t_2[C])$$

Many well-known examples of non-crisp FDs such as Similarity Dependencies [2], Differential Dependencies [43] or Neighborhood Dependencies [3] can be found in the literature and generally use a mix of metric predicates (metric associate with a threshold) and equality predicates, see also [9]. The substitution of equality by predicates considerably enlarges the spectrum of expression from string and numerical metrics to ordering constraints and probabilistic proximity. φ_{ncrisp} is an example of such an FD.

With crisp and non-crisp FDs, the computation of g_3 relies on the notion of *violating pairs*. A pair of tuples (t_1, t_2) is defined as a violating pair if it *does not satisfy* (or *violates*) φ. In the case of crisp FDs for instance, (t_1, t_2) is a violating pair (written $(t_1, t_2) \not\models \varphi$) iff $t_1[A] = t_2[A]$ for A in X and $t_1[C] \neq t_2[C]$. In other words, a violating pair is a contradiction

in the relation where two tuples have similar antecedents and dissimilar consequents with regard to the definition of φ. The set VP of violating pairs for a relation r and an FD φ can be written as follows:

$$\text{VP}(\varphi, r) = \{(t_1, t_2) | t_1, t_2 \in r, (t_1, t_2) \not\models_{nc} \varphi\} \tag{3}$$

Hence, g_3 is defined as the minimum proportion of tuples to be removed from relation r to get rid of all violating pairs [30]. Along with its direct derivatives the *error* and *confidence* (conf), g_3 can be expressed as follows:

$$\begin{aligned} g_3(\varphi, r) &= 1 - \frac{\max(|\{s | s \subseteq r, |\text{VP}(\varphi, s)| = 0\}|)}{|r|} \\ &= 1 - \frac{\max(|\{s | s \subseteq r, s \models_{nc} \varphi\}|)}{|r|} \\ &= \text{error}(\varphi, r) = 1 - \text{conf}(\varphi, r) \end{aligned}$$

We also define the *validation problem* versions of error and confidence introduced in [30] and formalized in [43]. These decision problems are often used to validate the veracity of a given FD automatically using a threshold chosen with domain experts (in FD mining algorithms for example). They can be expressed as follows:

- **Error validation problem** Given a threshold η_e, output YES if $\text{error}(\varphi, r) \leq \eta_e$ and NO otherwise.
- **Confidence validation problem** Given a threshold η_c, output YES if $\text{conf}(\varphi, r) \geq \eta_c$ and NO otherwise.

Thus, the error (or confidence) stands for the proportion of *bad* (or *good*) tuples and is meant to be minimized (or *maximized*). Despite one being simply the dual of the other, the computations of error and confidence differ in some situations, and in particular, we will see that their approximation guarantees are not the same in the case of non-crisp FDs. These differences in use as well as in computation justify making a clear distinction between them.

If g_3 can be computed in polynomial time in n with crisp FDs, relaxing the equality and losing its associated transitivity makes the decision version of error and confidence NP-Complete and their exact computation NP-Hard. The approach taken in this paper is inspired by [43] which converts error and confidence to graphs and solve them as the MVC and the Maximum Independent Set (MIS) respectively. However, the MVC is not only NP-Hard but also APX-complete and cannot be approximated with a better factor than 1.3606 [14] or even 2 if the unique games conjecture is true [29]. Its dual, the MIS, is even harder as it is strongly NP-Hard, Poly-APX-complete and cannot be approximated within a constant factor [18]. Despite these not very encouraging results, the many studies on hard problems have proposed solutions which are explored in this paper.

Multisets are distinguished from sets by the use of brackets ([ ]) in place of braces ({ }). Similarly to [12], [25], let an *equivalence class* r^x be the subset of r having x value on X such that $r^x = [t | t \in r, t[X] = x]$. Unless otherwise indicated, we use π_X in place of $\pi_X(r)$ with $\pi_X(r)$ the classical projection operator. Equally, the g_3, $error$ and $conf$ notations are used in place of $g_3(\varphi, r)$, $\text{error}(\varphi, r)$ and $\text{conf}(\varphi, r)$ respectively. Complexities are considered as tuple comparisons and not value comparisons. More generally, we consider $|X| + 1 << n$ with $|X|$ the number of antecedents.

Note that underlined appellations spread throughout the document will be used in the experiment section to reference specific algorithms. In particular, G3 *and* NCG3 *prefix algorithms regarding the computation of g_3 with crisp and non-crisp FDs respectively.*

III. Computing g_3 with Crisp FDs

In this section, we first propose multiple ways to compute g_3 exactly. After analyzing the guarantees of uniform random sampling, we then adapt and propose an improvement to an existing stratified sampling approach [11].

A. Computation overview

Let $\varphi : X \to C$ be a crisp FD and r a relation over R. In the case of crisp FDs, computing g_3 can be done in polynomial time in the number of rows n. For each equivalence class $r^x \in \pi_X$, there can only be one unique consequent to allow for the satisfaction of φ such that: $\max_{r^x \in \pi_X} |r^x[C]| = 1$. Thus, to keep the largest subset of tuples satisfying φ, it is sufficient to find the most frequent element in each equivalence class and discard all the other tuples, therefore removing all violating pairs. The normalized size of the discarded sets of tuples corresponds to g_3. In other words, the g_3 can be computed by finding all the groups of tuples sharing the same antecedent and counting the most frequent elements in their respective consequents.

B. Exact computation

GROUP-BY operations such as the one needed to find all equivalence classes r^x are generally sort- or hash-based with sorting optimizing memory and hashing time complexity. Both approaches are studied in the following and their full implementations are available on the GitHub repository.

Sorting: G3_MEMOPT first sorts the data and computes g_3 in one pass over the data. If the data is sorted externally and is then read in a streaming fashion to find each equivalence class, this method allows for low memory usage. In particular, the memory needed can be adjusted depending on the chunks used for external sorting with memory usage ranging from $\mathcal{O}(1)$ to $\mathcal{O}(n)$. The overall time complexity of $\mathcal{O}(n \log(n))$ is also bounded by the sorting operation.

Hashing: G3_TIMEOPT computes g_3 in one pass over the data by storing each equivalence class in a hash table. Hashing is less memory-efficient but allows for a lower theoretical time complexity of $\mathcal{O}(n)$. The theoretical memory needed is $\sum_{r^x \in \pi_X} |r^x[C]| = \mathcal{O}(n)$. However, significant memory and time complexity overheads might be required depending on the hashing algorithm.

C. Random sampling

For very large datasets, computing g_3 can become overwhelming in terms of time complexity and memory management. We present techniques for its scalability based on random sampling.

1) Uniform random sampling (G3_URS): We first consider the attractive approach of URS proposed in Algorithm 1. This simple approach is easy to implement, allows for massive gains in computation time and proposes good theoretical guarantees which are presented in Theorem III.1.

Algorithm 1 Uniform sampling g_3 (G3_URS)

Require:
Relation r, FD φ
Algorithm $\mathcal{A}$ such that $\mathcal{A}(\varphi, s) = g_3(\varphi, s)$
Confidence δ, Error ϵ
1: $m \Leftarrow \min(|r|, \left\lceil \frac{1}{2\epsilon^2} \ln(\frac{2}{1-\delta}) \right\rceil)$
2: $s \Leftarrow$ uniform random sample of size m drawn from r
3: **return** $\mathcal{A}(\varphi, s)$

Theorem III.1. *Let $\mathcal{A}(\varphi, r)$ be an algorithm computing the exact value of g_3 in $T_{\mathcal{A}}(n)$ time. Algorithm 1 computes an estimate $\hat{g}_3$ of g_3 such that $p(|\hat{g}_3 - g_3| \leq \epsilon) \geq \delta$ with confidence δ and error ϵ. Its time complexity is $\mathcal{O}(T_s(m,n) + T_{\mathcal{A}}(m))$ with $m = \min(n, \left\lceil \frac{1}{2\epsilon^2} \ln(\frac{2}{1-\delta}) \right\rceil)$ and $T_s(m,n)$ the complexity of sampling m of n tuples .*

The proof, based on Hoeffding's inequality [24], is omitted for brevity. Despite its elegance, this approach often performs badly owing to the many specificities of real datasets (very small equivalence classes, too many different consequents, etc., see Section V for experiments).

2) Advanced sampling schemes: We also examine advanced sampling schemes presented in [11] to compute the confidence of CFDs. Those strategies can almost be applied directly to crisp FDs as they are only a specific case of CFDs as explained in [11]. Among the proposed solutions, the 2-pass stratified random sampling (SRS) approach provided the best approximations in their experiments and is the one considered and implemented (G3_SRS) in this paper. Moreover, G3_SRS is implemented with [32] which proposes an improvement on classic reservoir sampling [47] which makes it possible to considerably speed up the first pass if the size of the dataset is known.

The G3_SRS algorithm proposes an estimate of the confidence in two passes over the data. In a *first pass*, it samples $t = 2\epsilon_c^{-2} log(\frac{2}{\delta})$ rows with reservoir sampling to provide an estimate $|\hat{r}^x|$ of the size $|r^x|$ of each equivalence class with error ϵ and confidence δ. In a *second pass*, it then samples z rows in each equivalence class r^x to compute an estimate $\hat{g}_3(\varphi, r^x)$ of $g_3(\varphi, r^x)$. Finally, it computes a weighted average to provide an estimate of $g_3(\varphi, r)$ for the full relation.

Note that we did not express the reservoir size z in the second pass. Indeed, two methods are proposed in [11] with regard to this: reservoir sampling and the space-saving algorithm [33]. In both cases, the chosen value for z does not take into account the individual equivalence class sizes which can result in extremely over-/under-estimated sample sizes. In her experiments, [33] uses a constant value of $z = 20$. However, while 20 can provide a good estimate for small equivalence class sizes, it is not able to estimate correctly the g_3 of very large ones (eg. 50000 tuples). On the opposite, choosing a large value such as 5000 performs better on large equivalence classes but also samples every element of smaller ones which results in a decline in performance. Therefore, we observe that estimating a good constant value for z is not possible as small groups and large groups can coexist in the same dataset. This is why propose to use a variable reservoir size z_x for each equivalence class r^x which depends on $|r^x|$ (improvement implemented in G3_SRSI). Notably, we use the estimate $|\hat{r}^x|$ made in the first pass for the size of each equivalence class r^x and then use Hoeffding's inequality with finite population

correction [41] to provide an adaptive sample size z_x such that:

$$z_x = \left\lceil (\frac{2\epsilon^2}{\ln(\frac{2}{1-\delta})} + \frac{1}{|\hat{r}^x|})^{-1} \right\rceil \tag{4}$$

with error ϵ and confidence δ. This solution brings two major improvements: nothing is to be assumed about the data beforehand to choose z manually, each reservoir size z_x is chosen so as to offer good guarantees on the approximation while sampling *just enough* tuples. We will see that this approach (G3_SRSI) works very well in practice and outperforms G3_SRS all the time.

IV. Computing g_3 with Non-crisp FDs

We now investigate the case of non-crisp FDs. We examine first the hardness of these problems and then approaches to compute error and confidence with non-crisp FDs, both exactly and with approximations. Note that in this section, error and confidence are examined independently to account for differences in their computing process and notably their approximations.

A. Hardness of error and confidence

In [43], it is proven that the error and confidence validation problems with differential dependencies (DDs) are NP-Complete by reductions from the MVC and the Clique [18] respectively.

Theorem IV.1. *The error and confidence validation problems are NP-Complete with non-crisp FDs.*

The proof is similar to [43] and is omitted. Note that DDs restrict the predicates to a class of distance metrics whose resulting values are compared with a threshold via an operator. Therefore, DDs are expressible as non-crisp FDs with a restriction on the predicates.

B. Computation overview

Let $\varphi : X \rightarrow C$ be a non-crisp FD and r a relation over R. We now formalize an equivalence making it possible to compute error and confidence with non-crisp FDs as a graph problem by solving the MVC and the MIS respectively. We derive an undirected graph $G_{\varphi,r} = (V, E)$ with V the set of vertices and E the set of edges such that:

$$V = \{t | t \in r\}, E = \text{VP}(\varphi, r)$$

Property IV.1. *Let r be a relation, a non-crisp FD φ and a graph $G_{\varphi,r} = (V, E)$. We have:*

$$\text{error}(\varphi, r) = \frac{|\text{MVC}(G_{\varphi,r})|}{|V|}, \text{conf}(\varphi, r) = \frac{|\text{MIS}(G_{\varphi,r})|}{|V|}$$

Proof: Let $C = \text{MVC}(G_{\varphi,r})$ be the result of the MVC. As a direct result from the MVC, there exists no edge e in E such that both endpoints are in $V \backslash C$ and therefore no violating pair in the corresponding set of tuples. Moreover, by definition, C is also the minimum set of vertices/tuples which can be removed to achieve this violating-pair-free set. Therefore, and as $|V| = n$, we prove the equivalence mentioned above:

$$\text{error}(\varphi, r) = \frac{|C|}{|V|} = \frac{|\text{MVC}(G_{\varphi,r})|}{|V|}$$

The proof is similar for the confidence. ■

Using the conversion described above, the computation of error and confidence is made up of two major operations:

1) **Violating pair enumeration** To convert the problem from a relation r and a non-crisp FD φ to a general undirected graph $G_{\varphi,r}$, we need to enumerate all violating pairs in r to create the edges. This operation is costly as it is quadratic in the number of tuples n and optimizations will be examined.
2) **Solving the MVC and the MIS** These two problems can be solved exactly in a reasonable amount of time for a *small* number of edges (number of violating pairs). However, their exponential complexity becomes quickly overwhelming and approximations must be examined. Two major types of approximations will be considered: approximation algorithms and sampling techniques.

Crisp FDs from a graph point of view: In the case of crisp FDs, and as they are only a special case of non-crisp FDs, the exact same conversion can be applied to compute error and confidence and thus g_3. In particular, for each equivalence class $r^x \in \pi_X$, the conversion generates an isolated *complete k-partite graph* made of $k = |r^x[C]|$ fully connected independent sets. This family of graphs is also a minimal superclass of P_4-tidy graphs in which the MVC and MIS have been proven to be solvable in linear time [19] which confirms our previous polynomial result.

C. Exact computation

1) Violating pair enumeration: Violating pair enumeration (VPE) consists in finding all pairs of tuples in a relation with similar antecedents and dissimilar consequents.

*Brute force (*VPE_BF*):* In the most general scenario, each tuple in r must be compared to all other tuples in a nested loop. With this scheme, $\mathcal{O}(T_n)$ *comparisons* are required with $T_n = \binom{n+1}{2} = \mathcal{O}(n^2)$. A *comparison* is the successive pairwise comparison of each attribute of two tuples which results in, at most, $|X| + 1$ computed predicates for each pair of tuples.

Hopefully, the problem of comparing pairs of records in a dataset has been extensively studied in multiple fields ranging from record linkage to similarity joins. Notably, some propositions allowing for significant time gains found in the literature can be applied and are described in the following. However, these potential optimizations are directly linked to the definition of the predicates and the attributes space and in some cases where no assumption can be made about either of them, T_n comparisons are required in practice. Moreover, storing all resulting violating pairs can be memory-intensive: for a dataset of 200k tuples, if all of them are in violation with all others and stored as *unsigned ints* pairs, $\sim$200GB are required to store them all.

*Blocking (*VPE_BLOCKOPT*):* The *blocking* indexing method is widely used in the field of record linkage as it allows for massive gains in time complexity (see [46] for a survey). It consists in grouping together similar values in attributes and processing each resulting block separately. As T_n is quadratic, processing smaller blocks will indeed reduce the overall complexity and can easily be combined with parallelization. In our case, this method is applicable only when the predicate is *equivalence relation* for an attribute in the *antecedents*. In general, most categorical attributes such as *zip codes* or *phone numbers* are usually compared using equality. Note that the gains of this method depend exclusively on the content of the attributes used for grouping as more distinct values lead to smaller blocks and therefore less processing time. Let $X_=$ be the set of *antecedents* with the *equality* predicate. After blocking into $B = |r[X_=]|$ blocs, the number of comparisons is bounded by $\mathcal{O}(B \cdot T_{n_{max}})$ with n_{max} the size of the largest block. Note that the blocking operation is similar to a GROUP-BY whose complexity is discussed in Section III.

*Attribute comparison order (*VPE_COMPOPT*):* There are situations in which no equivalence predicate is used in the antecedents or where blocking doesn't produce sufficiently small blocks for reasonable computation times. Thus, it is of interest to optimize the comparison inside the blocks. Notably, the order of the attributes checked successively plays an important role in the complexity of VPE: we want the attributes to be in an order which generates the fewest false positives. Indeed, if the comparison on the first attribute is positive, then the second one is checked and so on until it is either identified as a violating pair or rejected. However, if a pair of tuples is rejected by an attribute, all predicates computed before will have been a waste of resources by generating a false positive temporary violating pair (this is especially important for costly predicates such as the Levenshtein distance). We should therefore sort the attributes for comparison from the one generating the fewest violating pairs to the one generating the most. To evaluate the number of potential violating pairs of an attribute efficiently in the antecedents or in the consequents, we propose to perform VPE for each attribute on a sample of the datasets of predefined size. More precisely, we isolate each antecedent (eg. $A_0, A_1 \rightarrow C$ becomes $A_0 \rightarrow C$ to evaluate A_0) and then perform VPE on this modified FD. Finally, we sort the attributes in ascending order regarding their computed number of violating pairs and perform VPE on the full dataset.

*Candidate pairs in totally ordered space with monotonic predicate (*VPE_ORDEROPT*):* For further optimization regarding the comparison inside a block, we need to restrict the expressiveness of one predicate in the antecedents. The goal is to output a set of candidate pairs by first joining on an attribute for which the complexity can be optimized. Then, the full pairwise attribute comparison can be performed on this restricted number of candidates.

Notably, we consider the case where an antecedent A_i^o belongs to a *totally ordered* space. In particular, we consider monotonic symmetric predicates such that for elements $a, b, c \in dom(A_i^o)$, we have:

$$a \leq b \leq c, \phi_i(a, c) \implies \phi_i(a, b)$$

Many predicates using a monotone metric with a threshold found in the literature respect this condition but other predicates such as the one presented in Formula 2 in our hydropower running example also allow this optimization to be applied. In that very case, it is sufficient to sort the data in $\mathcal{O}(n \log(n))$ and apply a *sliding window algorithm* similar to the one proposed in [15] which can output all candidates in one pass over the sorted data. *Ages*, *incomes* and *sizes* are, except in very specific cases, ordered numerical attributes. Nonetheless, this method can still be very expensive when the number of candidate pairs is still large (notably when the dataset contains many violating pairs) and the time complexity is still bounded by $\mathcal{O}(T_n)$.

Note that other optimizations specific to some attributes and predicates could be achieved for faster VPE. The optimizations proposed in this section are meant to be generic and to provide leads for adapting the process to the specific cases one might encounter.

2) Solving error and confidence: Once the graph $G_{\varphi,r}$ resulting from VPE is constructed, error and confidence can be evaluated by solving MVC($G_{\varphi,r}$) and MIS($G_{\varphi,r}$) respectively. Two main types of algorithms exist for solving the MVC/MIS [6]: exact algorithms and heuristic algorithms. Exact algorithms guarantee the optimality of their solution but may take an unreasonable amount of time for large graphs. In contrast, heuristic algorithms such as local-search algorithms do not offer any guarantee but are known to propose in practice near-optimal solutions within a reasonable time (generally set by the user) without being limited by the size of the graph.

In FASTG3, we propose the exact solver WeGotYouCovered [22] (NCG3_EXACT), winner of PACE 2019, and the local search algorithm NuMVC [6] (NCG3_HEUR(t) with t the running time) which runs a certain time provided by the user and outputs the best solution found.

The special case of validation problems: In this specific case, it is possible to optimize the search space by using a fixed-parameter tractable (FPT) algorithm for the MVC for the error validation problem. The problem is therefore not to find a MVC but to answer the question: *Is there an MVC of size smaller than k?* For instance, [10] makes it possible to solve it in $\mathcal{O}(kn + 1.2738^k)$ with $k = n\eta_e$. Nonetheless, this algorithm has a klam value [16] of 190 (maximum value of k for which the algorithm is expected to be practical) which makes it suitable only for small thresholds η_e. As far as we know, the MIS does not have any FPT algorithm and the confidence validation problem is still intractable after this relaxation. More formally, it is W[1]-hard [16].

D. Approximation algorithms

Many approximation algorithms have been designed for the MVC and the MIS. Approximation algorithms generally express their guarantees by a ratio which can be constant or dependant on the problem parameters. We denote by C and S two instances of an MVC and an MIS for $G_{\varphi,r}$ and by $\tilde{C}$, and $\tilde{S}$ the result of an approximation algorithm for these same problems. Therefore, given a k-approximation algorithm for those problems with k the approximation ratio, we get:

$$|C| \leq |\tilde{C}| \leq k|C|,\ k^{-1}|S| \leq |\tilde{S}| \leq |S|$$

or equivalently

$$\text{error} \leq \tilde{\text{error}} \leq k\text{error},\ k^{-1}\text{conf} \leq \tilde{\text{conf}} \leq \text{conf}$$

For the MVC, a very well-known greedy algorithm developed by Gavril and Yanakakis and described in [38] achieves a 2-approximation ratio (NCG3_2APPROX) by computing the size of a maximal matching in the graph. This is the best constant-factor currently known for this problem. A more involved algorithm [28] achieves a better factor of $2-\Theta(\frac{1}{\sqrt{\log(V)}})$ but is dependant on the size of the graph. However, while a lower approximation factor is a guarantee for the stability of the algorithm, it does not always give the best results in practice. Hence, [13] proposes an experimental comparison of 6 approximation algorithms which only confirms this last statement. Notably, the Greedy Independent Cover (GIC) algorithm (NCG3_GIC) [21] appears to be the clear winner of this benchmark and is implemented in FASTG3. While it only provides an approximation ratio of at least $\frac{\sqrt{d}}{2}$ (with d the maximum degree of the graph), it often provides almost-perfect approximations on a wide variety of graphs in log-linear time. In this case, these algorithms are really fast to run and the VPE becomes the bottleneck of the operation. Therefore, computing multiple approximations and taking the smallest is a viable option as it requires VPE to be performed only once. For instance, NCG3_2APPROX can be used as a barrier for worst-case scenarios for NCG3_GIC.

Regarding the MIS, no constant-factor approximation algorithm is currently known. The most well-known algorithm for this problem is the minimum greedy algorithm presented in [21] from which the GIC algorithm presented above is derived. Its guarantee is an approximation factor of $\frac{d+2}{3}$. More generally, any upper-bound on the MVC acts as a lower-bound for the MIS without, however, the same approximation guarantees.

E. Random sampling

For very large datasets, random sampling is often a practical solution. We therefore now present sublinear algorithms able to estimate the size of the MVC or the MIS without looking at the entire graph. To explore a graph, they have only two options: (1) asking for the degree of a given vertex, (2) asking for the neighbors of a vertex.

1) Online VPE: To benefit from those algorithms, VPE has to be performed in an online fashion by fetching all tuples in violation with a given one *only* when needed. For a given tuple t, operations (1) and (2) correspond respectively to the *number* and the *list* of tuples in violation with t and are equivalent from an algorithmic point of view. The set of tuples in violation with a given tuple t is expressed as:

$$\text{VP}(\varphi, r, t) = \{t' | t' \in r, (t, t') \not\models \varphi\}$$

Fortunately, this can be done in a very optimized way without losing any optimization proposed in the Section IV-C1 by keeping all relevant information in memory (blocking, ordered attributes, type of join, etc.). The complexity of *online VPE* to retrieve all neighbors of a tuple t of r ranges from $|\text{VP}(\varphi, r, t)|$ to n depending on the available optimizations. It is therefore possible to propose a graph proxy hiding an on-the-fly VPE procedure to any sublinear graph algorithm, sparing time and memory in regard to complete VPE. This proxy is implemented in FASTG3.

2) Sublinear algorithms: As initially proposed in [39], most sublinear algorithms for the MVC work as follows: (1) sample a set of nodes from the graph, (2) decide for each if it belongs to an (approximate) MVC, (3) generalize the result to estimate the size of the MVC on the full graph. Following the observation used in NCG3_2APPROX that the size of an MVC is between the size and twice the size of any maximal matching, studies subsequent to [39] have proposed solutions to estimate the size of a matching in a graph and use it as bounds for the size of the MVC. In fine, this solution proposes a sublinear estimate of NCG3_2APPROX and therefore a 2-approximation of the size of the MVC. We notably implement [50] (NCG3_SUB09) and [37] (NCG3_SUB11) with respective query complexities of $\mathcal{O}(d^4/\epsilon^2)$ and $\mathcal{O}(\overline{d}^2 \cdot \text{poly}(1/\epsilon))$ (corrected from original paper following discussion with the authors) with d and $\overline{d}$ the maximum and average degrees. Following Hoeffding's inequality, by sampling $m = \min(n, \left\lceil \frac{1}{2\epsilon^2} \ln(\frac{2}{1-\delta}) \right\rceil)$ nodes, those algorithms provide an approximation $|\tilde{C}|$ of $|C|$ such that:

$$p(|C| - n\epsilon \leq |\tilde{C}| \leq 2|C| + n\epsilon) \geq \delta$$

V. Experiments

A. The FASTG3 Python Library

FASTG3 is an open-source library for computing the g_3 indicator with crisp and non-crisp FDs. This library is proposed as a Python module but achieves very good performances thanks to an underlying C++ implementation based on Cython. FASTG3 is available publicly on Github (github.com/datavalor/fastg3) and is used for all the following experiments. All benchmarks (also available on the repository) are run on the following configuration: *Ubuntu 20.04, Python 3.8, i7-7700k, 32GB of memory.*

All algorithms implemented and tested are summarized in Table II along with their complexities. VPE used to convert the computation of g_3 into a a graph problem in the case of non-crisp FDs also uses multiple algorithms and optimizations summarized in Table III.

TABLE II: Summary of all g_3 algorithms

FDs	Name[a]	Approx?	Complexity[abc]
Crisp	G3_MEMOPT	·	$\mathcal{O}(n \log(n))$
	G3_TIMEOPT	·	$\mathcal{O}(n)$
	G3_URS($\mathcal{A}$)	Yes	$T_{\mathcal{A}}(m)$
	G3_SRS [11]	Yes	$\mathcal{O}(n)$
	G3_SRSI	Yes	$\mathcal{O}(n)$
Non-crisp	VPE+NCG3_EXACT [22]	·	exponential
	VPE+NCG3_HEUR(t) [6]	Yes	$\mathcal{O}(n^2)$+timeout t
	VPE+NCG3_GIC [21]	Yes	$\mathcal{O}(n^2) + \mathcal{O}(n \log(n) + \|\text{VP}\|)$
	VPE+NCG3_2APPROX [38]	Yes	$\mathcal{O}(n^2) + \mathcal{O}(\|\text{VP}\| \log(n))$
	NCG3_SUB09[d] [50]	Yes	$\mathcal{O}(d^4/\epsilon^2) \cdot \mathcal{O}(n)$
	NCG3_SUB11[d] [37]	Yes	$\mathcal{O}(\overline{d}^2 \cdot \text{poly}(1/\epsilon)) \cdot \mathcal{O}(n)$

[a] VPE+ prefixes algorithm which require to convert the input table to a graph by using VPE composed of $|VP|$ violating pairs/edges in $\mathcal{O}(n^2)$.
[b] d an $\overline{d}$ denote for the maximum and average degree of the VPE graph.
[c] ϵ denotes for the statistical error parameter of the algorithms.
[d] NCG3_SUB09 and NCG3_SUB11 use online VPE which multiplies the number of queries made by the algorithms by $\mathcal{O}(n)$.

TABLE III: Summary of all VPE algorithms

Name	Attribute space	Predicate type
VPE_BF	Any	Any
VPE_BLOCKOPT	Any	Equality
VPE_COMPOPT	Any	Any
VPE_ORDEROPT	Totally ordered	Monotonic

B. Datasets

Two real-life datasets and one synthetic one are used for the experiments:

- **Diamonds** This public dataset is composed of 53,940 tuples with 9 categorical and numerical attributes describing various properties of a set of diamonds.
- **Hydroturbine** Composed of 511,017 tuples, this dataset is similar r_{toy} with 6 numerical attributes describing various properties of a water turbine. This dataset cannot be made public for industrial reasons.
- **Syn** We also propose the use of a generator of synthetic datasets making it possible to choose multiple parameters related to the computation of g_3 with crisp FDs. It is defined as $Syn(g = 0.5, n = 1M, e = 300, a = 2, c = 1, u = 0)$ with known g_3 value (g) as well as variable numbers of tuples (n), equivalence classes (e), antecedents (a), and consequents (c) and percentage of unique consequents in each equivalence class while keeping the target g_3 achievable (u). We use the default values in the following experiments when parameters are not defined. x means that the parameter is currently being tested (eg. $Syn(g = x, n = 100)$).

As long as the number of antecedents or consequents is not changed in the experiments, the default FDs presented in the following sections are used. The predicates used in the non-crisp section correspond to uncertainties devised with domain experts for Hydroturbine and are estimated for Diamonds as it is a public dataset. The FDs themselves correspond to potential functions one may encounter, for example, in a prediction problem.

C. Crisp functional dependencies

1) Settings: The FDs used in this section for Diamonds and Hydroturbine are respectively:

- $carat, cut, color, clarity, depth \rightarrow price$
  - 41,350 equivalence classes, $g_3 = 0.20$
- $flow, opening, position \rightarrow power$
  - 354,867 equivalence classes, $g_3 = 0.13$

Unless otherwise indicated, sampling algorithms use confidence $\delta = 0.95$ and error $\epsilon = 0.01$. This corresponds to $\min(18445, n)$ tuples sampled in the first pass of G3_SRS and G3_SRSI as well as the sample size used by G3_URS. G3_URS is used in conjunction with G3_TIMEOPT. If G3_SRSI chooses the reservoir size in the second pass using Formula 4 (with $\delta = 0.95$ and $\epsilon = 0.05$ in our experiments), a value needs to be chosen for G3_SRS. We follow the original paper [11] and use a constant value of $z = 100$ unless stated otherwise. This value is larger than the $z = 20$ that they used originally but it seemed fairer regarding the variety of our datasets.

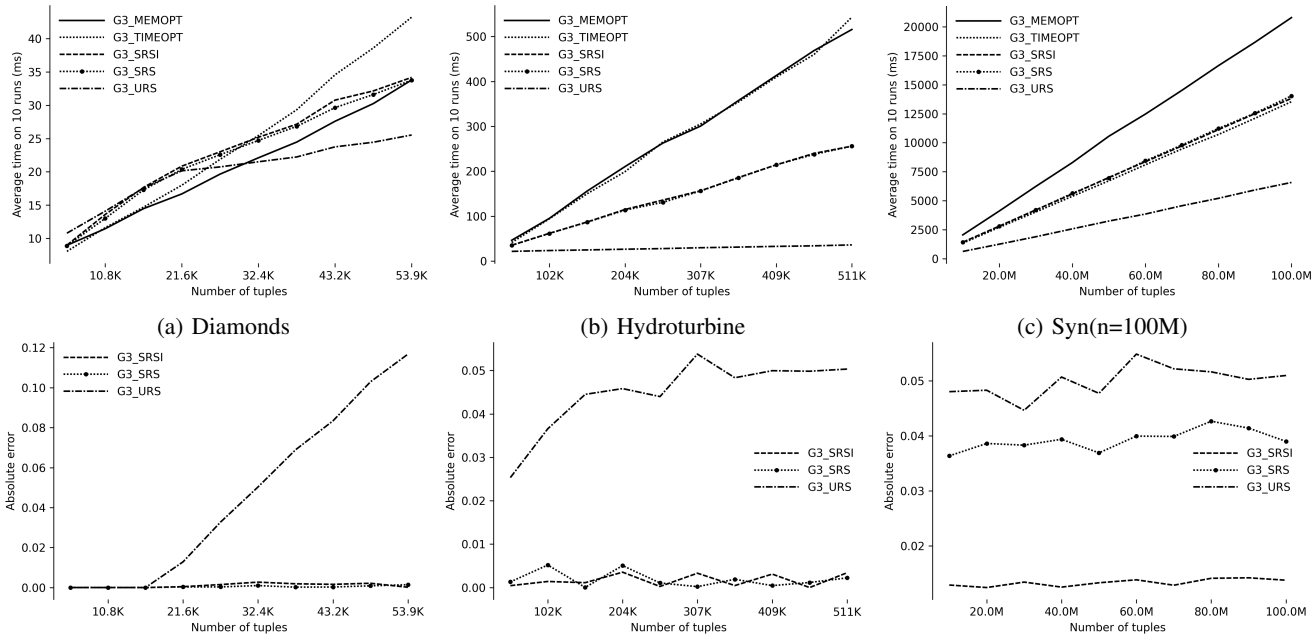
2) Results: Figure 1 presents the effect of the number of tuples on the time performance and the approximation accuracy. We observe that for small datasets such as Diamonds, the sorting operation is not an issue and that G3_TIMEOPT is not better than G3_MEMOPT. Interestingly, they achieve the same performances on the larger dataset Hydroturbine and that G3_TIMEOPT beats G3_MEMOPT largely on $Syn(n = 100M)$. The reasons for this are the large equivalence classes and the low number of unique consequents of Syn which allow the hashing algorithm to reallocate memory less often. Figure 1c shows the linear scalability of our solutions for very large datasets (tests up until $n = 100M$). As shown in Figure 2, the number of antecedents has an important effect on the running time. Indeed, the tuple-to-tuple equality check used in every algorithm (hashing, sorting, etc.) becomes longer as the number of attributes' values to be compared grows ($\sim$ linear effect).

For random sampling, we observe that G3_SRSI and G3_SRS have approximately the same running times. They are generally not competitive for small datasets such as Diamonds owing to their computation overheads (notably reservoir sampling) but become efficient for large ones. We see that G3_SRSI is always at least as accurate as G3_SRS and often proposes near-optimal approximations. In Figure 1f, we notably observe one of the drawbacks of the original G3_SRS: with only 300 equivalence classes, the reservoir size of 100 becomes insufficient which considerably deteriorates the approximation quality. In general, G3_URS is really fast but is only usable with large equivalence classes. Indeed, when the size of the equivalence classes grows (i.e. their number decreases), the proportion of tuples sampled in each one increases, leading to greater accuracy.

Finally, Figure 3 presents the effect of the various parameters of the Syn dataset on the approximation accuracy of random sampling algorithms. It confirms that G3_URS provides poor approximations for datasets with small equivalence classes and therefore requires a large number of samples. However, G3_SRS and G3_SRSI work really well with a small number of sampled tuples. In Figure 3e, we also observe that algorithms tend to provide a worse approximate for g_3 over 0.5. Indeed, at that point, the most frequent element in each equivalence class is not in *strict majority* which is known to be a hard case in *frequent elements problems* (for instance, this is the limit of exact *heavy hitters* approaches). Finally, we can see that

having very few distinct elements in each equivalence class also degrades the approximation as the most frequent element becomes harder to distinguish from the others.

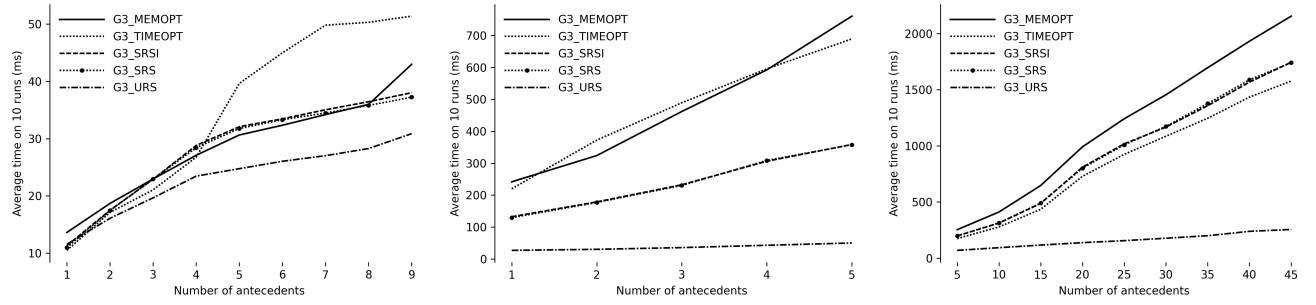
To summarize, G3_MEMOPT and G3_TIMEOPT perform both well with a slight advantage for G3_MEMOPT when there are few equivalence classes or unique consequents. For random sampling, G3_SRSI is to be preferred in most cases and its confidence and error in both passes could be decreased further to reduce its execution time. Finally, g_3 values under 0.5 and with more unique consequents are better approximated.



(a) Diamonds (b) Hydroturbine (c) Syn(n=100M)

(d) Diamonds (e) Hydroturbine (f) Syn(n=100M)

Figure 1: Influence of the number of tuples on time and approximation performances with crisp FDs.



(a) Diamonds (b) Hydroturbine (c) Syn(a=x)

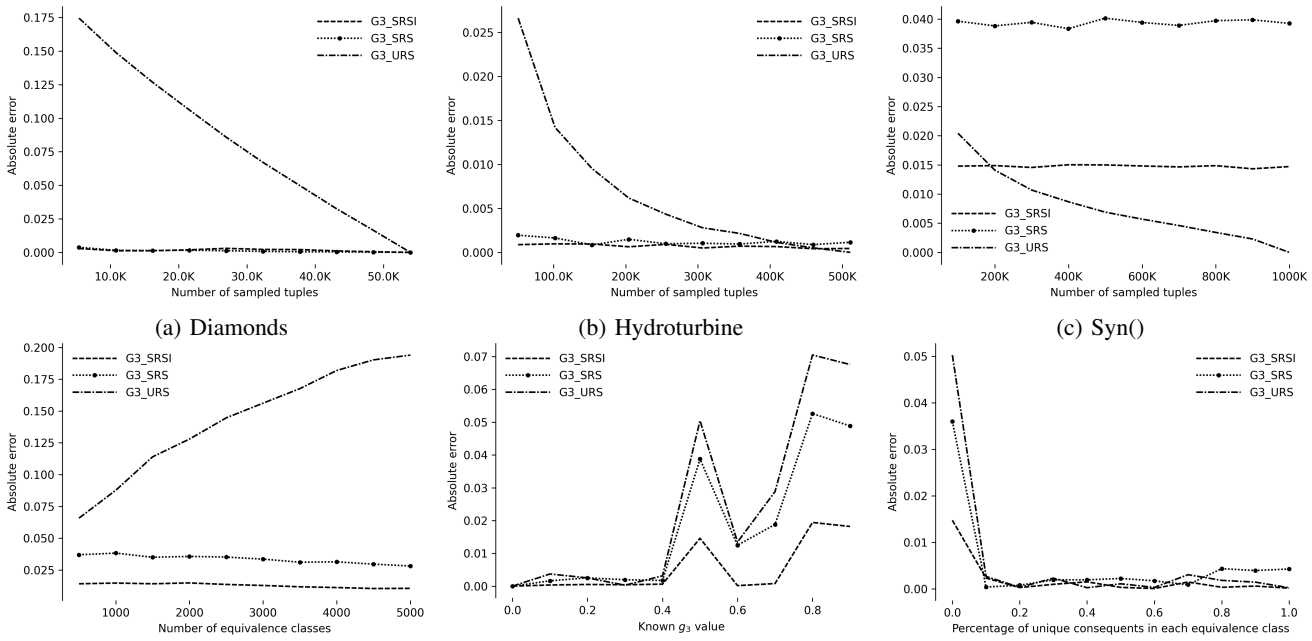
Figure 2: Influence of the number of antecedents on time performances with crisp FDs.

D. Non-crisp functional dependencies

1) Settings: The FDs used in this section for Diamonds and Hydroturbine are respectively:

- $[carat \pm 0.05], [x \pm 0.05], [y \pm 0.05], [z \pm 0.05], [depth \pm 0.05], [depth \pm 0.05], cut, color, clarity \rightarrow [price \pm 10]$
  - 21,182 violating pairs, $g_3 = 0.22$
- $[flow \pm 0.05], [opening \pm 0.03], [elevation \pm 0.03] \rightarrow [power \pm 0.05]$
  - 2,972,255 violating pairs, $g_3 = 0.31$

The Syn dataset is not used in this section as it has been designed specifically for crisp FDs. Hydroturbine is also reduced to a subset of 200,000 rows to keep reasonable computation times, notably to compute the exact MVC with NCG3_EXACT.



(a) Diamonds (b) Hydroturbine (c) Syn()

(d) Syn(e=x) (e) Syn(g=x) (f) Syn(u=x)

Figure 3: Influence of multiple parameters on the approximation performances with crisp FDs.

NuMVC is used with a constant running time of 1 second (NCG3_HEUR(1s)) and is therefore not shown on the time performance graphs. Unless otherwise indicated, 2000 tuples/nodes are sampled for the two sublinear algorithms NCG3_SUB09 and NCG3_SUB11. We also focus solely on the error for concision. Nonetheless, most experimental conclusions also apply to confidence.

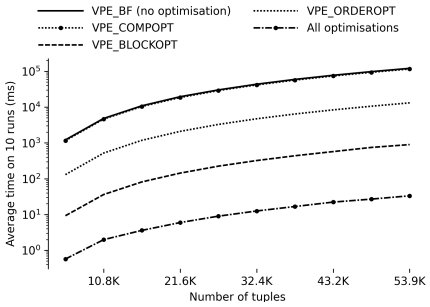


Figure 4: Violating pairs enumeration on the Diamonds dataset with various levels of optimizations.

2) Results:

Violating pair enumeration: Figure 4 presents the execution time of VPE with different levels of optimization for the Diamonds dataset. All these levels are achievable because the non-crisp FD considered contains categorical antecedents with the equality predicate ($cut, color, clarity$) as well as at least one totally ordered antecedent with monotonic predicate (one of $carat, x, y, z, depth, depth$). If blocking is so effective, it is because the projection of Diamonds on $\{cut, color, clarity\}$ generates very small blocks which can be processed efficiently. With all optimizations combined, very reasonable computation times are achieved. On the other hand, only VPE_COMPOPT and VPE_ORDEROPT can be used for Hydroturbine and the high number of rows and especially the high number of violating pairs make the process still more time consuming with $\sim 30s$ total for processing the 200k rows (graph not shown here for brevity).

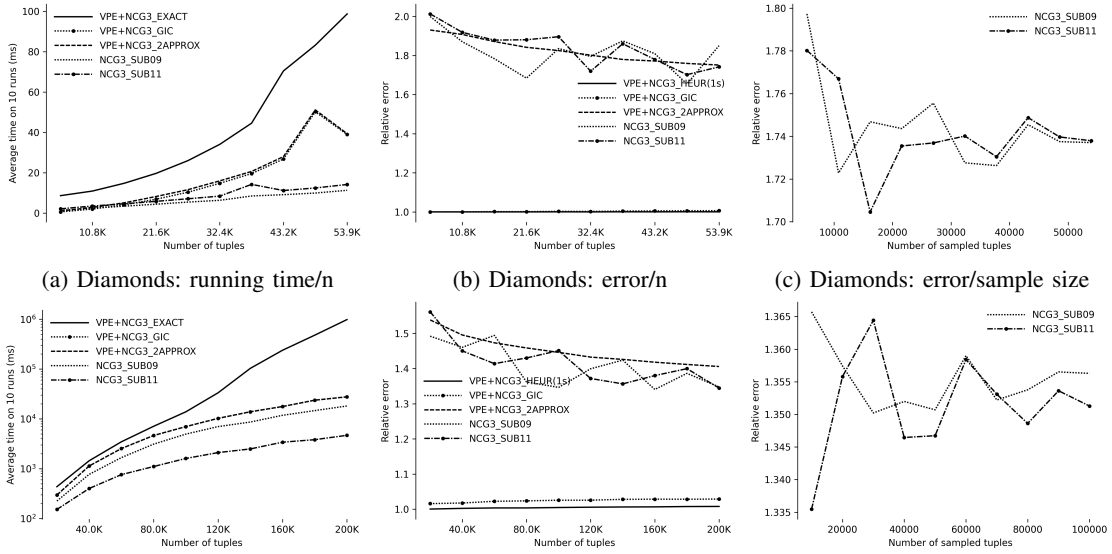
To summarize, the number of violating pairs is by far the most limiting factor. In addition to more violating pairs meaning more *required* comparisons, this is also likely to generate more potential *false positives* (pairs of tuples which are compared to finally conclude that they are not a violating pair) and therefore predicates computed in vain. Therefore, it is possible

to perform optimized VPE on very large datasets with very few violating pairs but it may also be very long with medium datasets which contain a lot of them.

Computing g_3/error: Figure 5 presents the time and approximation performances of the computation of error with non-crisp FDs. We observe that NCG3_GIC offers excellent approximation accuracy. NCG3_HEUR(1s) provides perfect results in constant 1s time which is especially useful for very large graphs with many edges where NCG3_EXACT takes too much time. In all our tests, there is no case where NCG3_2APPOX is preferred and it is, in general, closer to its 2 approximation ratio guarantee than the exact value.

Sublinear algorithms offer significant time performance benefits by replacing full VPE by online VPE. We can see that both are almost equivalent with Diamonds when NCG3_SUB11 performs better than NCG3_SUB09. Indeed, the cubic guarantees of NCG3_SUB11 are likely to perform better that the quartic guarantees of NCG3_SUB09. It is also reassuring to see that their approximation is always very close to NCG3_2APPROX of which they initially propose an estimate. Moreover, we can see that they do not require a large sample size to work well.

To summarize, NCG3_EXACT performs well but becomes limited when the dataset contains numerous violating pairs. Nonetheless, NCG3_GIC proposes a fast a accurate estimate of the error almost as competitive as NCG3_HEUR. If VPE becomes too long, sublinear algorithms propose a good estimate of the NCG3_2APPROX in reasonable time, even with a small sample size. In general, NCG3_SUB11 is preferred over NCG3_SUB09.



(a) Diamonds: running time/n (b) Diamonds: error/n (c) Diamonds: error/sample size

(d) Hydroturbine: running time/n (e) Hydroturbine: error/n (f) Hydroturbine: error/sample size

Figure 5: Influence of multiple parameters on the time of approximation performances with non-crisp FDs.

3) Summary of experiments: With crisp FDs, the computation of g_3 is dependent on the number of tuples and is therefore highly scalable up to millions of tuples. Nevertheless, approximation algorithms can be used to avoid iterating over every tuple and therefore speed up the computation. G3_URS appeared to be insufficient but stratified approaches propose an efficient alternative (G3_SRS), especially when used with dynamic reservoir size in the second size (G3_SRSI) to account for various equivalence class sizes. These approximation algorithms are especially effective for smaller g_3 values.

With non-crisp FDs, the computation of g_3 is dependent on the number of tuples but also on the number of violating pairs. When VPE can be achieved in reasonable time (eg. many optimizations can be applied), G3_EXACT can be used to compute the error exactly for a small number of violating pairs and G3_GIC or G3_HEUR(t) propose efficient approximation alternatives. When VPE becomes too long, sublinear algorithms (especially G3_SUB11) offer a quicker alternative at the expense approximation quality.

VI. Related Work

The g_3 indicator is not the only known coverage measure. For example, purity dependencies [42] are based on an impurity measure, soft FDs [26] and probabilistic FDs [48] use a probabilistic approach when the alternative of a compression-based measure is preferred in partial determinations [40]. An overview of various coverage measures is presented in [7]. Nonetheless, the g_3 indicator is undoubtedly among the most widely used coverage measures: Approximate FDs [30], approximate DDs

[43] or approximate comparable dependencies [44] are examples of FDs using the g_3 indicator as their coverage measure and many mining algorithms also use the g_3 indicator to find FDs which almost hold in a relation [30], [49], [25], [8]. This ubiquity of the g_3 indicator is notably due its intuitive interpretation and its flexibility. When the g_3 only requires the definition of FD satisfiability (which is at the core of any FD definition), the equality relaxation generalized by non-crisp FDs is harder to capture with coverage measures such as the probabilistic ones mentioned above where the need to group values together struggles with the loss of transitivity.

Crisp FDs: In [30] introducing g_3 in the context of FD mining, the computation process is described but no detailed algorithm for its exact computation is considered. [25] presents a simple algorithm to compute the g_3-error for a given equivalence class. Concerning random sampling, an approach is proposed in [30] for the validation problems. Notably, it is proven that a uniform sample of size at least $O(\frac{\sqrt{n}}{\eta_e}\log(\frac{1}{\delta}))$ is required to solve it with probability δ. [11] explores the computation of the confidence of conditional FDs by proposing streaming algorithms based on advanced sampling schemes. One of their propositions has been implemented (G3_SRS), improved (G3_SRSI) and compared to other alternatives.

Non-crisp FDs: g_3 has been extensively used in the context of FD mining but only little work for computing it with non-crisp FDs has been carried out. [43] established for the first time an equivalence between the error (or confidence) to the MVC (or MIS) in the case of DDs. A well-known 2-approximation algorithm [38] is used to solve the the MVC in both [43] and [8]. This algorithm corresponds to NCG3_2APPROX in this paper and has been shown to provide average results in practice. The alternative of sublinear algorithms has been studied and a simple algorithm from [39] is proposed. Significant improvements in terms of complexity and approximation guarantees have been proposed in this field and [35], [50], [37] (we study [50], [37] in this paper).

Nonetheless, the problem of converting the input from a relation to a graph problem is not considered by [43] nor [8]. This is, however, a computationally intensive process which has proven during our experiments to be a bottleneck in regard to some of the very efficient approximation algorithms used to solve the MVC/MIS. The VPE approach used to achieve such conversion is a very interesting problem which is at the crossroads of record linkage and similarity joins. If we tried to cover most of the relevant literature, the case of unordered metric space has been deliberately omitted for brevity, despite the fact that it is often useful for comparing, for example, strings with metrics such as the Levenshtein distance. This operation is similar to a *range self-similarity join in a metric space* and has been extensively studied [51] through the use of indexing such as tree data structures (see [23] for a survey) or divide-and-conquer algorithms such as QuickJoin [27]. In any case, the pair-wise tuple enumeration is known to be a hard problem with no silver bullet. Equally, while efficient approximation algorithms exist, solving the MVC exactly in the most general case still remains a major computational challenge.

Finally, while measuring the veracity of a function in a dataset has been the subject of this paper, analyzing violating pairs to understand why it behaves in this way is also of interest. Presented in [17], the web application ADESIT uses FASTG3 to display indicators such as g_3 and to propose an interactive visualization of violating pairs in the lens of supervised learning or any function in general.

VII. Conclusion and Future Work

In this paper, we studied scalable techniques to compute the g_3 indicator with crisp FDs (known to be polynomial) and non-crisp FDs (known to be NP-Hard). For crisp FDs and for very large datasets, advanced sampling schemes were proposed and their theoretical guarantees were analyzed. For non-crisp FDs, two subproblems were identified. First, for the enumeration of the violating pairs which is quadratic in the size of the dataset, blocking techniques, attribute ordering and ordered spaces were studied. Second, an analysis of available approximate algorithms and recent developments in sublinear algorithms were examined for solving the MVC.

All the considered algorithms were then implemented and tested through extensive experiments. For crisp FDs, the propositions were fairly scalable while keeping a good approximation accuracy for sampling approaches. Progressive sampling techniques could be considered to speed up the algorithms. For non-crisp FDs, we observed that the bottleneck of the computation lies in the violating pair enumeration process. Sublinear algorithms offer important time savings but they all simulate NCG3_2APPOX which is known to provide average approximations in practice. Thus, as future work, it could be interesting to adapt a practically better approximation algorithm (eg. degree-based heuristics) which should provide better results.

Acknowledgments

We would like to thank Graham Cormode and Divesh Srivastava for the quality of our exchange on their paper [11]. Thanks also to Krzysztof Onak for kindly responding to our questions on [37]. Finally, our thanks go to Datavalor initiative at INSA Lyon for funding a part of this work.

REFERENCES

[1] ARMSTRONG, W. W. Dependency structures of data base relationships. In *IFIP congress* (1974), vol. 74, Geneva, Switzerland, pp. 580–583.

[2] BAIXERIES, J., KAYTOUE, M., AND NAPOLI, A. Computing similarity dependencies with pattern structures. In *The Tenth International Conference on Concept Lattices and their Applications-CLA 2013,* (2013), pp. 33–44.

[3] BASSÉE, R., AND WIJSEN, J. Neighborhood dependencies for prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2001), Springer, pp. 562–567.

[4] BEHNEL, S., BRADSHAW, R., CITRO, C., DALCIN, L., SELJEBOTN, D. S., AND SMITH, K. Cython: The best of both worlds. *Computing in Science & Engineering 13*, 2 (2011), 31–39.

[5] BOHANNON, P., FAN, W., GEERTS, F., JIA, X., AND KEMENTSIETSIDIS, A. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd international conference on data engineering* (2007), IEEE, pp. 746–755.

[6] CAI, S., SU, K., LUO, C., AND SATTAR, A. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research 46* (2013), 687–716.

[7] CARUCCIO, L., DEUFEMIA, V., AND POLESE, G. Relaxed functional dependencies—a survey of approaches. *IEEE Transactions on Knowledge and Data Engineering 28*, 1 (2015), 147–165.

[8] CARUCCIO, L., DEUFEMIA, V., AND POLESE, G. On the discovery of relaxed functional dependencies. In *Proceedings of the 20th International Database Engineering & Applications Symposium* (2016), pp. 53–61.

[9] CHARDIN, B., COQUERY, E., PAILLOUX, M., AND PETIT, J.-M. Rql: a query language for rule discovery in databases. *Theoretical Computer Science 658* (2017), 357–374.

[10] CHEN, J., KANJ, I. A., AND XIA, G. Improved parameterized upper bounds for vertex cover. In *International symposium on mathematical foundations of computer science* (2006), Springer, pp. 238–249.

[11] CORMODE, G., GOLAB, L., FLIP, K., MCGREGOR, A., SRIVASTAVA, D., AND ZHANG, X. Estimating the confidence of conditional functional dependencies. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2009), SIGMOD '09, Association for Computing Machinery, p. 469–482.

[12] COSMADAKIS, S. S., KANELLAKIS, P. C., AND SPYRATOS, N. Partition semantics for relations. *Journal of Computer and System Sciences 33*, 2 (1986), 203–233.

[13] DELBOT, F., AND LAFOREST, C. Analytical and experimental comparison of six algorithms for the vertex cover problem. *Journal of Experimental Algorithmics (JEA) 15* (2010), 1–1.

[14] DINUR, I., AND SAFRA, S. On the hardness of approximating minimum vertex cover. *Annals of mathematics* (2005), 439–485.

[15] DOHNAL, V., GENNARO, C., AND ZEZULA, P. Similarity join in metric spaces using ed-index. In *International Conference on Database and Expert Systems Applications* (2003), Springer, pp. 484–493.

[16] DOWNEY, R. G., AND FELLOWS, M. R. *Parameterized complexity*. Springer Science & Business Media, 2012.

[17] FAURE--GIOVAGNOLI, P., PETIT, J.-M., SCUTURICI, V.-M., AND LE GUILLY, M. ADESIT: Visualize the Limits of your Data in a Machine Learning Process. In *International Conference on Very Large Data Bases* (Copenhaguen, Denmark, Aug. 2021), pp. 2679–2682.

[18] GAREY, M. R., AND JOHNSON, D. S. Computers and intractability. *A Guide to the* (1979).

[19] GIAKOUMAKIS, V., ROUSSEL, F., AND THUILLIER, H. On p_4-tidy graphs. *Discrete Mathematics and Theoretical Computer Science 1* (1997), 17–41.

[20] GOLAB, L., KARLOFF, H., KORN, F., SRIVASTAVA, D., AND YU, B. On generating near-optimal tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment 1*, 1 (2008), 376–390.

[21] HALLDÓRSSON, M. M., AND RADHAKRISHNAN, J. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica 18*, 1 (1997), 145–163.

[22] HESPE, D., LAMM, S., SCHULZ, C., AND STRASH, D. Wegotyoucovered: The winning solver from the pace 2019 challenge, vertex cover track. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing* (2020), SIAM, pp. 1–11.

[23] HJALTASON, G. R., AND SAMET, H. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS) 28*, 4 (2003), 517–580.

[24] HOEFFDING, W. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[25] HUHTALA, Y., KÄRKKÄINEN, J., PORKKA, P., AND TOIVONEN, H. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal 42*, 2 (1999), 100–111.

[26] ILYAS, I. F., MARKL, V., HAAS, P., BROWN, P., AND ABOULNAGA, A. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), pp. 647–658.

[27] JACOX, E. H., AND SAMET, H. Metric space similarity joins. *ACM Transactions on Database Systems (TODS) 33*, 2 (2008), 1–38.

[28] KARAKOSTAS, G. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms (TALG) 5*, 4 (2009), 1–8.

[29] KHOT, S., AND REGEV, O. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences 74*, 3 (2008), 335–349.

[30] KIVINEN, J., AND MANNILA, H. Approximate inference of functional dependencies from relations. *Theoretical Computer Science 149*, 1 (1995), 129–149.

[31] LE GUILLY, M., PETIT, J., AND SCUTURICI, V. Evaluating classification feasibility using functional dependencies. *Trans. Large Scale Data Knowl. Centered Syst. 44* (2020), 132–159.

[32] LI, K.-H. Reservoir-sampling algorithms of time complexity o (n (1+ log (n/n))). *ACM Transactions on Mathematical Software (TOMS) 20*, 4 (1994), 481–493.

[33] METWALLY, A., AGRAWAL, D., AND EL ABBADI, A. Efficient computation of frequent and top-k elements in data streams. In *International conference on database theory* (2005), Springer, pp. 398–412.

[34] NAMBIAR, U., AND KAMBHAMPATI, S. Mining approximate functional dependencies and concept similarities to answer imprecise queries. In *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004* (2004), pp. 73–78.

[35] NGUYEN, H. N., AND ONAK, K. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science* (2008), IEEE, pp. 327–336.

[36] NOVELLI, N., AND CICCHETTI, R. Functional and embedded dependency inference: a data mining point of view. *Information Systems 26*, 7 (2001), 477–506.

[37] ONAK, K., RON, D., ROSEN, M., AND RUBINFELD, R. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (2012), SIAM, pp. 1123–1131.

[38] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[39] PARNAS, M., AND RON, D. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science 381*, 1-3 (2007), 183–196.

[40] PFAHRINGER, B., AND KRAMER, S. Compression-based evaluation of partial determinations. In *KDD* (1995), pp. 234–239.

[41] SERFLING, R. J. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics* (1974), 39–48.

[42] SIMOVICI, D. A., CRISTOFOR, D., AND CRISTOFOR, L. Impurity measures in databases. *Acta Informatica 38*, 5 (2002), 307–324.

[43] SONG, S. *Data dependencies in the presence of difference*. PhD thesis, Hong Kong University of Science and Technology, 2010.

[44] SONG, S., CHEN, L., AND PHILIP, S. Y. Comparable dependencies over heterogeneous data. *The VLDB journal 22*, 2 (2013), 253–274.

[45] SONG, S., GAO, F., HUANG, R., AND WANG, C. Data dependencies extended for variety and veracity: A family tree. *IEEE Transactions on Knowledge and Data Engineering* (12 2020).

[46] STEORTS, R. C., VENTURA, S. L., SADINLE, M., AND FIENBERG, S. E. A comparison of blocking methods for record linkage. In *International conference on privacy in statistical databases* (2014), Springer, pp. 253–268.

[47] VITTER, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS) 11*, 1 (1985), 37–57.

[48] WANG, D. Z., DONG, X. L., SARMA, A. D., FRANKLIN, M. J., AND HALEVY, A. Y. Functional dependency generation and applications in pay-as-you-go data integration systems. In *WebDB* (2009).

[49] WANG, Y., SONG, S., CHEN, L., YU, J. X., AND CHENG, H. Discovering conditional matching rules. *ACM Transactions on Knowledge Discovery from Data (TKDD) 11*, 4 (2017), 1–38.

[50] YOSHIDA, Y., YAMAMOTO, M., AND ITO, H. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the forty-first annual ACM symposium on Theory of computing* (2009), pp. 225–234.

[51] ZEZULA, P., AMATO, G., DOHNAL, V., AND BATKO, M. *Similarity search: the metric space approach*, vol. 32. Springer Science & Business Media, 2006.