Dscaler: Synthetically Scaling A Given Relational Database

J.W. Zhang
School of Computing
National University of Singapore
jiangwei@u.nus.edu

Y.C. Tay
School of Computing
National University of Singapore
dcstayyc@nus.edu.sg

ABSTRACT

The Dataset Scaling Problem (DSP) defined in previous work states: *Given an empirical set of relational tables $\mathcal{D}$ and a scale factor s, generate a database state $\widetilde{\mathcal{D}}$ that is similar to $\mathcal{D}$ but s times its size.* A DSP solution is useful for application development ($s < 1$), scalability testing ($s > 1$) and anonymization ($s = 1$). Current solutions assume all table sizes scale by the same ratio s.

However, a real database tends to have tables that grow at different rates. This paper therefore considers *non-uniform scaling* (nuDSP), a DSP generalization where, instead of a single scale factor s, tables can scale by different factors.

DSCALER is the first solution for nuDSP. It follows previous work in achieving similarity by reproducing correlation among the primary and foreign keys. However, it introduces the concept of a *correlation database* that captures fine-grained, per-tuple correlation.

Experiments with well-known real and synthetic datasets $\mathcal{D}$ show that DSCALER produces $\widetilde{\mathcal{D}}$ with greater similarity to $\mathcal{D}$ than state-of-the-art techniques. Here, similarity is measured by number of tuples, frequency distribution of foreign key references, and multi-join aggregate queries.

1. INTRODUCTION

In the Dataset Scaling Problem (DSP), the input is an empirical relational database state $\mathcal{D}$ and a positive real number s that specifies the ***scale factor***; the output is a synthetic database state $\widetilde{\mathcal{D}}$ that is similar to $\mathcal{D}$ but s times its size. The similarity can be measured by graph properties, query results, transaction throughput, etc.

Take, for example, the dataset $\mathcal{D}$ = `DoubanBook`[1] generated by a Chinese social network that allows the creation and sharing of content related to books. A query like "average rating of those users who have commented on a book" should return similar values for $\mathcal{D}$ and for a scaled $\widetilde{\mathcal{D}}$.

[1]https://book.douban.com/

Proceedings of the VLDB Endowment, Vol. 9, No. 14

The motivation for DSP depends on s. A start-up with a small $\mathcal{D}$ may want a larger version $\widetilde{\mathcal{D}}$ for testing the scalability of their system architecture. Since $s > 1$ for this example, $\widetilde{\mathcal{D}}$ is necessarily artificial.

On the other hand, an enterprise with a large $\mathcal{D}$ may want a smaller version $\widetilde{\mathcal{D}}$ for application development. Since $s < 1$, one can sample $\widetilde{\mathcal{D}}$ from $\mathcal{D}$, but it is not obvious how similarity is to be enforced. For example, $\widetilde{\mathcal{D}}$ must satisfy the same foreign key (FK) constraints and have the same probability distributions as $\mathcal{D}$. To draw an analogy, if we randomly sample nodes/edges from a directed graph G, the sampled $\widetilde{G}$ may differ from G in terms of connectedness, correlation between in/out-degree, etc. This is one reason for $\widetilde{\mathcal{D}}$ to be synthetic, instead of being a sample of $\mathcal{D}$.

Another reason for $\widetilde{\mathcal{D}}$ to be artificial lies in data protection, which is necessary if, say, $\mathcal{D}$'s owner provides $\widetilde{\mathcal{D}}$ to a vendor. If $\widetilde{\mathcal{D}}$ is a sample from $\mathcal{D}$, the personal and proprietary data in $\widetilde{\mathcal{D}}$ can be hidden through anonymization. However, there are well-known examples where such efforts have failed [12, 22]. In contrast, no anonymization is necessary if $\widetilde{\mathcal{D}}$ is synthetic. Nonetheless, some information leakage is unavoidable since $\widetilde{\mathcal{D}}$ is, after all, similar to $\mathcal{D}$; e.g., one might deduce from $\widetilde{\mathcal{D}}$ the users' gender ratio in $\mathcal{D}$.

Data protection is also why the case $s = 1$ can be useful: $\widetilde{\mathcal{D}}$ is the same size as $\mathcal{D}$, but all of $\widetilde{\mathcal{D}}$'s data are artificial.

The scale factor s can be interpreted in different ways. It can be GBytes, or the number of users if $\mathcal{D}$ is from a social network, etc. The first version of DSP [29] assumes uniform scaling, i.e. the number of tuples for each table in $\mathcal{D}$ scales by the same ratio s. This is a strong assumption: If $s > 1$ and $\widetilde{\mathcal{D}}$ models the growth of $\mathcal{D}$, the tables may grow at different rates. For example, the number of customers for a retailer may grow faster than the number of suppliers.

We therefore introduce a generalization of DSP to **Non-Uniform Dataset Scaling Problem (nuDSP)**:

> Given a database of relational tables $\mathcal{D} = \{T_1, \ldots, T_K\}$ and a ***scaling vector*** $\mathbf{s} = [s_{T_1}, \ldots, s_{T_K}]$, generate a similar database $\widetilde{\mathcal{D}} = \{\widetilde{T_1}, \ldots, \widetilde{T_K}\}$ where $|\widetilde{T_i}| = s_{T_i}|T_i|$ and $|T|$ is the number of tuples in T.

This paper presents a solution to nuDSP, namely DSCALER.

1.1 Approaches that do not solve nuDSP

For scaling down (i.e. $s_{T_i} < 1$), one natural approach is to sample the tuples [17, 30]. Such a solution is nontrivial (e.g. consider the issue of FK constraints mentioned above),

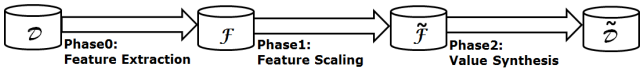

Figure 1: The 3-phase DSCALER Scaling Framework, where there are multiple possibilities for $\mathcal{F}$.

but sampling does not work for scaling up (i.e. $s_{T_i} \geq 1$) anyway. For scaling up, one possible approach is to make s_{T_i} copies of table T_i [7], but that means a table T for an `Employee-Manager` management tree, say, will scale to some $\widetilde{T}$ with s_T *separate* management trees; besides, this `copy&paste` approach does not work for scaling down.

If similarity between $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ is to be measured with a set of queries $\mathcal{Q}$, one approach may be to use $\mathcal{Q}$ when generating $\widetilde{\mathcal{D}}$ [3, 4]. However, this will mean regenerating $\widetilde{\mathcal{D}}$ whenever $\mathcal{Q}$ is changed. In contrast, the UpSizeR solution to DSP [29] does not target any queries, but generates $\widetilde{\mathcal{D}}$ by replicating the data correlations it extracts from $\mathcal{D}$. However, UpSizeR only solves the special case $s_{T_1} = \cdots = s_{T_K} = s$.

1.2 Main ideas and insights

DSCALER has 3 phases, as illustrated in Figure 1:
Phase0: Extract some features $\mathcal{F}$ from $\mathcal{D}$.
Phase1: Use vector $\mathbf{s}$ to scale $\mathcal{F}$ to $\widetilde{\mathcal{F}}$.
Phase0: Use $\widetilde{\mathcal{F}}$ to generate tuples and thus synthesize $\widetilde{\mathcal{D}}$.
The main ideas and insights in this solution are as follows:

(I1) Features. Abstractly speaking, UpSizeR has the same 3 phases in Figure 1. This 3-phase process is thus a *framework*, in the sense that other nuDSP solutions can use it with a different choice for $\mathcal{F}$ and different algorithms in each phase. The feature set $\mathcal{F}$ plays two roles in this framework:

(i) $\mathcal{F}$ is an input to constructing $\widetilde{\mathcal{D}}$ (via $\widetilde{\mathcal{F}}$).

(ii) $\mathcal{F}$ defines the similarity between $\mathcal{D}$ and $\widetilde{\mathcal{D}}$.

These two roles are conflicting: The smaller $\mathcal{F}$ is, the easier it is to construct $\widetilde{\mathcal{D}}$; on the other hand, the bigger $\mathcal{F}$ is, the more similar are $\mathcal{D}$ and $\widetilde{\mathcal{D}}$.

(I2) Correlation. To choose the feature set $\mathcal{F}$, we observe that, fundamentally, relational queries rely on joins to reconstruct tuples that are decomposed into separate tables, and these joins mostly match key values. For $\widetilde{\mathcal{D}}$ to be similar to $\mathcal{D}$ in query results, $\widetilde{\mathcal{D}}$ and $\mathcal{D}$ must therefore have similar correlation among their primary and foreign key values. This is why DSCALER chooses key value correlation as the features to include in $\mathcal{F}$.

(I3) Uniqueness. One kind of correlation that complicates value synthesis for keys is *uniqueness*. For example, in the `DoubanBook` social network, the table for books in $\widetilde{\mathcal{D}}$ should not have two tuples with the same `title-author` pair for these FKs. Yet, if the scale factor for the table is big enough, then (by Pigeonhole Principle) repetition is unavoidable. We therefore need to determine necessary and sufficient conditions for uniqueness to be possible.

(I4) Decoupling. There are two keywords in nuDSP: *scaling* and *similar*. The DSCALER solution decouples these two requirements into Phase1 and Phase2. Phase1 (scaling) determines the number of tuples to generate for each correlation pattern, and Phase2 (similarity) constructs the necessary number of tuples. These two phases are independent: one can change the algorithm in one phase without affecting the algorithms in the other phase.

(I5) *CoDa*. The features in $\mathcal{F}$ can be statistical distributions, graph properties, etc. To capture the correlation in (**I2**), we view $\mathcal{D}$ as a directed graph induced by the FK constraints, and base $\mathcal{F}$ on the frequency distributions for indegrees and outdegrees in this graph. These reference counts are per-tuple (in contrast to the coarse granularity used by UpSizeR, which groups tuples into clusters and capture inter-cluster correlation). The frequency distributions can be represented as tables, so $\mathcal{F}$ is itself a relational database that we call the ***correlation database***, denoted $\mathcal{F} = CoDa_{\mathcal{D}}$. Hence, Phase1 first scales $CoDa_{\mathcal{D}}$ to give another correlation database $\widetilde{\mathcal{F}} = CoDa_{\widetilde{\mathcal{D}}}$, before Phase2 generates key values to transform $CoDa_{\widetilde{\mathcal{D}}}$ into $\widetilde{\mathcal{D}}$.

(I6) Size. Recall from (**I1**) that $\mathcal{F}$ plays the role of input to constructing $\widetilde{\mathcal{D}}$. With $\mathcal{F}$ defined as $CoDa_{\mathcal{D}}$ **(I5)**, this input is much smaller than $\mathcal{D}$ and easier to store and transfer. For example, an enterprise can upload $\mathcal{F}$ into the cloud, then use it to generate $\widetilde{\mathcal{D}}$ in the cloud.

(I7) Loss. On the other hand, recall from (**I1**) that $\mathcal{F}$ also defines similarity between $\mathcal{D}$ and $\widetilde{\mathcal{D}}$. With $\mathcal{F} = CoDa_{\mathcal{D}}$, any correlation that is not captured by $CoDa_{\mathcal{D}}$ is lost in constructing $\widetilde{\mathcal{D}}$. In a social network, for example, two friends are more likely to comment on each other's post, but $CoDa_{\mathcal{D}}$ does not include such correlations.

1.3 Overview

To summarize, our contribution in this paper are:

1. We generalize the DSP definition to nuDSP by introducing a table scaling vector $\mathbf{s} = [s_{T_1}, \ldots, s_{T_K}]$ to reflect non-uniform table growth in real datasets.
2. We present DSCALER, a solution to nuDSP. It is the first solution for non-uniform scaling.
3. A naive scaling algorithm can violate the uniqueness requirement (**I3**) above. We present necessary and sufficient conditions for satisfying uniqueness.
4. We compare DSCALER to state-of-the-art techniques, using synthetic (`TPC-H`) and real (`financial`, `DoubanBook`) datasets. The results show DSCALER generates greater similarity between $\widetilde{\mathcal{D}}$ and $\mathcal{D}$, as measured by syntactic (number of tuples, degree distributions) and semantic (aggregate queries) metrics.

We begin by surveying related work in Section 2. Section 3 introduces some necessary definitions and notation. Feature extraction for $CoDa_{\mathcal{D}}$ is straightforward, so we will not elaborate on Phase0. Phase1 and Phase2 can be broken down into smaller steps, as shown in Figure 2. These steps are described in detail in Section 4 for Phase1 and Section 5 for Phase2.

For the experiments, Section 6 describes the alternative algorithms we compare DSCALER with, specifies the similarity measures that we use, and summarizes the results. Section 7 and Section 8 present the results for uniform and non-uniform scaling, respectively. Section 9 revisits the issue of information loss (**I7**), before we conclude the paper in Section 10.

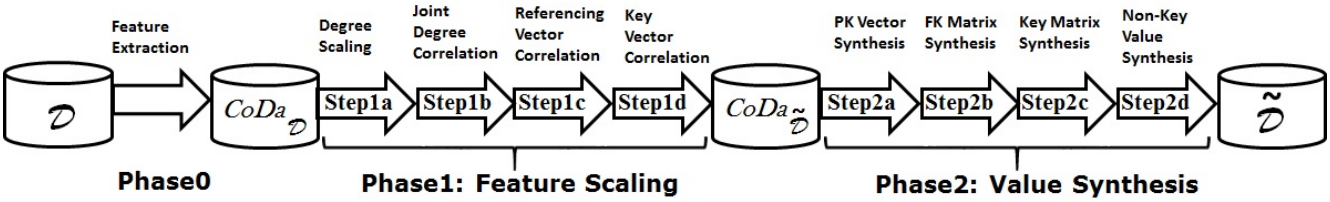

Figure 2: Phase1 and Phase2 in Figure 1 can be broken down into smaller steps.

2. RELATED WORK

In Section 1.1, we pointed to some previous work when describing possible approaches to solving nuDSP. We now survey these and other related work.

Related work on generating relational datasets have a variety of objectives; e.g. some focus on environment or software testing [26, 27]. They can be classified into two categories: query-aware and query-independent.

Query-aware techniques [3, 4] usually need a set of queries as input for dataset generation. For example, RQP (Reverse Query Processing) [4] takes a set of queries and the database schema as input, and generates a synthetic database as output. Similarly, QAGen [5] uses a set of queries and some cardinality constraints as input to generate a synthetic database that satisfies the constraints. However, for n different queries, QAGen will generate n different databases. Later, MyBenchmark [19] tackles this QAGen issue by minimizing the number of databases generated for one set of queries. Nonetheless, MyBenchmark does not always generate a single database. These query-aware techniques provide similarities that are based on the input queries.

Query-independent techniques [6, 14, 26, 27, 29] do not have a set of queries as input, and aim to provide similarity for all queries (as is the case for DSCALER). Examples include DGL (Data Generation Language) [6] and Houkjær et al.'s graph model [16].

UpSizeR is another graph-based tool [29] that uses attribute correlation extracted from an empirical database to generate a synthetic database; it is the first solution to DSP. ReX [7] is later work that scales up the original database by an integer factor s, using an automated representative extrapolation technique. This is a variant of `copy&paste` that generates s copies of $\mathcal{D}$ (hence the integer value for s). For example, given $\mathcal{D}$ for a social network and $s = 3$, ReX would produce 3 separate social networks. Moreover, the technique cannot scale down a dataset.

Database sampling is a form of scaling down [17, 30]. These sampling algorithms have two main motivations: (i) to reduce the computational cost for data mining [18, 24], and (ii) to provide approximate answers [2, 9] for queries. For example, a density-biased sampling approach [23] can be used to find the same clusters as in the original database, and thus reduce the cost for data mining. For (ii), Join Synopses [1] and Linked Bernoulli Synopses [13] are examples. However, maintaining the foreign key integrity for these synopses is computationally expensive, hence time consuming for database generation. VFDS [8] is a more recent tool that maintains the referential integrity in the sampling. The authors' experiments show that VFDS has similar accuracy as Join Synopses and Linked Bernoulli Synopses, and performs at least 300 times faster.

A	B		C		
PK	PK	FK_A	PK	FK_A	FK_B
a_1	b_1	a_2	c_1	a_1	b_1
a_2	b_2	a_2	c_2	a_2	b_1
	b_3	a_1	c_3	a_1	b_2
	b_4	a_1	c_4	a_2	b_3

Figure 3: Input database $\mathcal{D}_{eg}$ in our running example.

A	B		C		
PK	PK	FK_A	PK	FK_A	FK_B
(2, 2)	(2)	(2, 2)	(0)	(2, 2)	(2)
(2, 2)	(1)	(2, 2)	(0)	(2, 2)	(2)
	(1)	(2, 2)	(0)	(2, 2)	(1)
	(0)	(2, 2)	(0)	(2, 2)	(1)

Figure 4: $CoDa_{\mathcal{D}_{eg}}$. Each value v in T of Figure 3 is replaced by joint degree $\Delta_T(v)$.

For anonymization without scaling, Gupta et al. employ privacy mechanisms [21] to hide key statistics of the original database for privacy control [15]. This approach is later extended [20], with a guarantee of privacy for individuals in the database. For DSCALER, any privacy loss in key values will be via $CoDa_{\mathcal{D}}$ which, however, has no values from $\mathcal{D}$.

Scaling appears in other fields as well. The AO benchmark [11] is the first tool that scales down an RDF dataset. Later, RBench [25] is proposed to both scale down and up. However, these two benchmarks are evaluated with different metrics (*dataset coherence, relationship specialty, literal diversity*), so it would be unfair to use them for comparison.

None of these papers provide a solution for nuDSP — they either do not scale the dataset, or cannot scale by nonuniform factors. For experimental comparison, VFDS and UpSizeR are the state-of-the-art algorithms that are closest to DSCALER, so we will further describe them in Section 6.

3. DEFINITIONS AND NOTATION

To simplify the definitions, notation and description of DSCALER, we assume that *if T' references T (denoted $T' \to T$), it does so via one foreign key constraint only.* This assumption can be easily relaxed for DSCALER.

We use the database $\mathcal{D}_{eg}$ in Figure 3 as a running example. In $\mathcal{D}_{eg}$, table B references table A, and table C's foreign keys FK_A and FK_B are the primary keys (PKs) of A and B respectively. For any tuple t with PK value p, we refer to t as p if there is no ambiguity; e.g. we use a_2 to represent the second tuple in A. The following definition narrows the scope of the feature set $\mathcal{F}$ to the keys:

DEFINITION 1. *For a table T, the **FK matrix** $\mathbb{F}_T$ is the projection of T on its foreign keys, and the **key matrix** $\mathbb{K}_T$ is the projection of T on its primary and foreign keys.*

In Section 4.3, we will consider the possibility that this projection yields repeated rows in $\mathbb{F}_T$. We can view each tuple as a node in a graph, with edges pointing from PK tuples to FK tuples. We thus get the following:

DEFINITION 2. *Suppose $T' \to T$ and $t \in T$. The **degree** of t, denoted $\delta_{T' \to T}(t)$, is the number of tuples in T' referencing t.*

In $\mathcal{D}_{eg}$, $\delta_{B \to A}(a_2) = 2$, $\delta_{C \to B}(b_2) = 1$, and $\delta_{C \to B}(b_4) = 0$. To capture the correlation among FKs in a table, we need their joint distribution:

DEFINITION 3. *Suppose tables $T_1, \ldots, T_K$ reference T and $t \in T$. Then $\Delta_T(t) = (\delta_{T_1 \to T}(t), \ldots, \delta_{T_K \to T}(t))$ is the **joint degree** for t.*

In $\mathcal{D}_{eg}$, $\Delta_A(a_1) = (2, 2)$, $\Delta_B(b_1) = (2)$ and $\Delta_B(b_2) = (1)$. These joint distributions form the feature set $\mathcal{F}$:

DEFINITION 4. *The **Correlation Database** of $\mathcal{D}$, denoted $CoDa_{\mathcal{D}}$, is the database that omits non-key values and substitutes $\mathcal{D}$'s key values with their joint degrees.*

$CoDa_{\mathcal{D}_{eg}}$ is shown in Figure 4. The correlation patterns among keys are modeled as vectors:

DEFINITION 5. *For table T and $t \in T$, the **referencing vector (RV)** $\phi_T(t)$ is the projection of t in $CoDa_{\mathcal{D}}$ on its foreign keys. The **key vector (KV)** $\Phi_T(t)$ is the projection of t in $CoDa_{\mathcal{D}}$ on its primary and foreign keys.*

In $\mathcal{D}_{eg}$, $\phi_C(c_1) = ((2, 2), (2))$, $\phi_C(c_3) = ((2, 2), (1))$, $\Phi_B(b_1) = ((2), (2, 2))$ and $\Phi_C(c_1) = ((0), (2, 2), (2))$. DSCALER needs to count these patterns:

DEFINITION 6. *Suppose $T' \to T$, $t \in T$ and $\Delta_T(t) = \alpha$. The **appearance number** $\Upsilon_{T' \to T}(\alpha)$ is the number of times α appears in the foreign key of T' in $CoDa_{\mathcal{D}}$.*

For $\mathcal{D}_{eg}$, $\Upsilon_{C \to A}((2, 2)) = 4 = \Upsilon_{B \to A}((2, 2))$. Lastly, the ***frequency distributions*** for degree (O_δ), joint degree (O_Δ), RV (O_ϕ), and KV (O_Φ) count the various patterns:

- $O_\delta^{T' \to T}(x) = w$ means w tuples in T have degree x, and these w tuples are referenced from T'.
- $O_\Delta^T(\alpha) = w$ means w tuples in T have joint degree α.
- $O_\phi^T(\beta) = w$ means w tuples in T have RV β.
- $O_\Phi^T(\gamma) = w$ means w tuples in T have KV γ.

In $\mathcal{D}_{eg}$, $O_\delta^{B \to A}(2) = 2$, $O_\delta^{C \to B}(1) = 2$, $O_\Delta^B((1)) = 2$, $O_\Delta^B((0)) = 1$, $O_\phi^C((2, 2), (1)) = 2$, $O_\phi^B((2, 2)) = 4$, $O_\Phi^B((1), (2, 2)) = 2$ and $O_\Phi^C((0), (2, 2), (1)) = 2$.

Table 1 lists the above notation for easy reference.

In the following sections, we will use $\mathcal{D}_{eg}$ to demonstrate how it is scaled to $\widetilde{\mathcal{D}_{eg}}$ in Figure 5. The scaling vector is $\mathbf{s} = [s_A, s_B, s_C] = [2, \frac{7}{4}, 2]$.

4. PHASE1: FEATURE SCALING

Phase0 for feature extraction is straightforward, so we proceed directly to Phase1 for feature scaling, which has 4 steps (Figure 2): **Step1a** uses the scaling vector $\mathbf{s}$ to scale the degree distributions $O_\delta^{T' \to T}$; **Step1b** combines the degrees (scalars) to derive the joint-degree distributions $O_\Delta^{\widetilde{T}}$; **Step1c** combines the joint degrees into vectors that capture the FK correlation $O_\phi^{\widetilde{T}}$; and **Step1d** combines the FK correlation with PK degrees to get $O_\Phi^{\widetilde{T}}$. Phase1 transforms $CoDa_{\mathcal{D}_{eg}}$ in Figure 4 to $CoDa_{\widetilde{\mathcal{D}_{eg}}}$ in Figure 6.

Notation	Description
$\mathcal{D}_{eg}/\widetilde{\mathcal{D}_{eg}}$	original/scaled example database
s_T	scaling factor for table T
$T/\widetilde{T}$	original/scaled table
PK/FK	primary/foreign Key
$\mathbb{F}/\mathbb{K}$	FK matrix/Key matrix
$\delta_{T' \to T}(t)$	number of tuples in T' referencing t.
$\Delta_T(t)$	joint degree for $t \in T$
$\Upsilon_{T' \to T}(\alpha)$	appearance number for α in T'
$\phi_T(t)$	referencing vector (RV) for t in T
$\Phi_T(t)$	key vector(KV) for t in T
O_δ, O_Δ O_ϕ, O_Φ	frequency distribution for degree, joint degree, RV, KV
$\alpha/\widetilde{\alpha}$	original/scaled joint degree
$\beta/\widetilde{\beta}$	original/scaled RV
$\gamma/\widetilde{\gamma}$	original/scaled KV

Table 1: Frequently Used Notation

$\widetilde{A}$	$\widetilde{B}$		$\widetilde{C}$		
PK	PK	FK_A	PK	FK_A	FK_B
$\widetilde{a_1}$	$\widetilde{b_1}$	$\widetilde{a_2}$	$\widetilde{c_1}$	$\widetilde{a_2}$	$\widetilde{b_1}$
$\widetilde{a_2}$	$\widetilde{b_2}$	$\widetilde{a_3}$	$\widetilde{c_2}$	$\widetilde{a_2}$	$\widetilde{b_2}$
$\widetilde{a_3}$	$\widetilde{b_3}$	$\widetilde{a_2}$	$\widetilde{c_3}$	$\widetilde{a_3}$	$\widetilde{b_1}$
$\widetilde{a_4}$	$\widetilde{b_4}$	$\widetilde{a_3}$	$\widetilde{c_4}$	$\widetilde{a_4}$	$\widetilde{b_2}$
	$\widetilde{b_5}$	$\widetilde{a_4}$	$\widetilde{c_5}$	$\widetilde{a_3}$	$\widetilde{b_3}$
	$\widetilde{b_6}$	$\widetilde{a_4}$	$\widetilde{c_6}$	$\widetilde{a_4}$	$\widetilde{b_4}$
	$\widetilde{b_7}$	$\widetilde{a_1}$	$\widetilde{c_7}$	$\widetilde{a_1}$	$\widetilde{b_5}$
			$\widetilde{c_8}$	$\widetilde{a_1}$	$\widetilde{b_6}$

Figure 5: Scaled $\widetilde{\mathcal{D}_{eg}}$ for $[s_A, s_B, s_C] = [2, \frac{7}{4}, 2]$

$\widetilde{A}$	$\widetilde{B}$		$\widetilde{C}$		
PK	PK	FK_A	PK	FK_A	FK_B
(1, 2)	(2)	(2, 2)	(0)	(2, 2)	(2)
(2, 2)	(2)	(2, 2)	(0)	(2, 2)	(2)
(2, 2)	(1)	(2, 2)	(0)	(2, 2)	(2)
(2, 2)	(1)	(2, 2)	(0)	(2, 2)	(2)
	(1)	(2, 2)	(0)	(2, 2)	(1)
	(1)	(2, 2)	(0)	(2, 2)	(1)
	(0)	(1, 2)	(0)	(1, 2)	(1)
			(0)	(1, 2)	(1)

Figure 6: Scaled $CoDa_{\widetilde{\mathcal{D}_{eg}}}$

4.1 Step1a: Degree Scaling

DSCALER first uses Algorithm 1 to scale the degree frequency distribution $O_\delta^{T' \to T}$ to $O_\delta^{\widetilde{T'} \to \widetilde{T}}$, with 3 requirements: (i) $O_\delta^{\widetilde{T'} \to \widetilde{T}}$ should be similar to $O_\delta^{T' \to T}$, (ii) $\sum_x O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) = |\widetilde{T}|$ and (iii)$\sum_x O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) \times x = |\widetilde{T'}|$. The details are as follows:

Tuple Scaling. Let $S_x = O_\delta^{T' \to T}(x) \times s_T$ for all x. Then, $O_\delta^{T' \to T}$ is first scaled to $O_\delta^{\widetilde{T'} \to \widetilde{T}}$ by the following operation:

$$O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) = \begin{cases} \lceil S_x \rceil & \text{with probability } S_x - \lfloor S_x \rfloor \\ \lfloor S_x \rfloor & \text{with probability } \lceil S_x \rceil - S_x \end{cases} \quad (1)$$

Thus, the average is $\mathrm{E}[O_\delta^{\widetilde{T'} \to \widetilde{T}}(x)] = S_x = O_\delta^{T' \to T}(x) \times s_T$.

Tuple Adjustment. With randomness introduced in Eq.1, we may have $\widetilde{n} = \sum_x O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) \neq |\widetilde{T}|$. Hence, $|\widetilde{n} - |\widetilde{T}||$ tuples with random degrees are added/removed, to make $\sum_x O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) = |\widetilde{T}|$.

FK Adjustment. *Tuple Adjustment* enforces $\widetilde{n} = |\widetilde{T}|$. Let $\widetilde{m} = \sum_x O_\delta^{\widetilde{T'} \to \widetilde{T}}(x) \times x$. Next, we need to scale the number of referencing tuples to make $\widetilde{m} = |\widetilde{T'}|$:

Case $\widetilde{m} < |\widetilde{T'}|$: increase the number of high degree tuples and decrease the number of low degree tuples in $\widetilde{T}$.

Case $\widetilde{m} > |\widetilde{T'}|$: decrease the number of high degree tuples and increase the number of low degree tuples in $\widetilde{T}$.

Algorithm 1: Degree Scaling

```
Tuple Scaling
Tuple Adjustment
while m̃ ≠ |T̃'| do
    FK Adjustment
return O_δ^{T̃'→T̃}
```

$\widetilde{B} \rightarrow \widetilde{A}$
$O_\delta^{\widetilde{B}\rightarrow\widetilde{A}}(1) = 1$
$O_\delta^{\widetilde{B}\rightarrow\widetilde{A}}(2) = 3$

$\widetilde{C} \rightarrow \widetilde{A}$
$O_\delta^{\widetilde{C}\rightarrow\widetilde{A}}(2) = 4$

$\widetilde{C} \rightarrow \widetilde{B}$
$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(0) = 1$
$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(1) = 4$
$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(2) = 2$

Figure 7: Degree Frequency Distribution for $\widetilde{\mathcal{D}_{eg}}$

$\widetilde{A}$
$O_\Delta^{\widetilde{A}}(1,2) = 1$
$O_\Delta^{\widetilde{A}}(2,2) = 3$

$\widetilde{C}$
$O^{\widetilde{C}}{}_\Delta(0) = 8$

$\widetilde{B}$
$O_\Delta^{\widetilde{B}}(0) = 1$
$O_\Delta^{\widetilde{B}}(1) = 4$
$O_\Delta^{\widetilde{B}}(2) = 2$

Figure 8: Joint Degree Frequency Distribution for $\widetilde{\mathcal{D}_{eg}}$

To illustrate, $CoDa_{\mathcal{D}_{eg}}$ in Figure 4 has

$$O_\delta^{C\rightarrow B}(0) = 1,\ O_\delta^{C\rightarrow B}(1) = 2,\ O_\delta^{C\rightarrow B}(2) = 1$$

Tuple scaling with $s_B = \frac{7}{4}$ gives

$$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(0) = 2,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(1) = 4,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(2) = 2$$

Thus, $\widetilde{n} = 2+4+2 \neq |\widetilde{B}| = s_B|B| = 7$; tuple adjustment removes 1 tuple with random degree (degree= 2), so

$$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(0) = 2,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(1) = 4,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(2) = 1$$

Next, $\widetilde{m} = (0\times2)+(1\times4)+(2\times1) = 6 < |\widetilde{C}| = s_C|C| = 8$, so FK adjustment increases the number of high degree tuples (degree=2), and decrease the number of low degree tuples (degree=0), to give

$$O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(0) = 1,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(1) = 4,\ O_\delta^{\widetilde{C}\rightarrow\widetilde{B}}(2) = 2$$

The scaled degree frequency distribution after FK adjustment is presented in Figure 7.

4.2 Step1b: Joint Degree Correlation (JDC)

Recall from Definition 3 that, for table T and $t \in T$, the joint degree is $\Delta_T(t) = (\delta_{T_1\rightarrow T}(t), \ldots, \delta_{T_K\rightarrow T}(t))$ where tables $T_1, \ldots, T_K$ reference T. Having scaled the degree distribution from $O_\delta^{T_1{}'\rightarrow T}$ to $O_\delta^{\widetilde{T_1}'\rightarrow\widetilde{T}}$, we need to combine them to construct the joint-degree distribution $O_\Delta^{\widetilde{T}}$, with the aim of preserving the joint-degree correlation in $\mathcal{D}$ (Figure 2).

Algorithm 2 sketches the algorithm for constructing $O_\Delta^{\widetilde{T}}$ from $O_\delta^{\widetilde{T_1}'\rightarrow\widetilde{T}}$. Intuitively, suppose $\widetilde{T_1}$ and $\widetilde{T_2}$ reference $\widetilde{T}$; to synthesize a tuple $\widetilde{t} \in \widetilde{T}$ with joint degree $(\widetilde{d_1}, \widetilde{d_2})$, Algorithm 2 chooses a tuple $\widetilde{t_1}$ from $O_\delta^{\widetilde{T_1}\rightarrow\widetilde{T}}$ with $\delta_{\widetilde{T_1}\rightarrow\widetilde{T}}(\widetilde{t_1}) = \widetilde{d_1}$ and $\widetilde{t_2}$ from $O_\delta^{\widetilde{T_2}\rightarrow\widetilde{T}}$ with $\delta_{\widetilde{T_2}\rightarrow\widetilde{T}}(\widetilde{t_2}) = \widetilde{d_2}$. It then combines $\widetilde{t_1}$ and $\widetilde{t_2}$ to form a new tuple $\widetilde{t}$, so $\widetilde{t}$ has joint degree $(\widetilde{d_1}, \widetilde{d_2})$.

In detail, for any table $\widetilde{T}$, Algorithm 2 loops through $O_\Delta^T(x_1, \ldots, x_k)$ and executes the following steps:

1-Norm Minimization. For each $(x_1, \ldots, x_k)$, choose $(\widetilde{x_1}, \ldots, \widetilde{x_k})$ **closest** to $(x_1, \ldots, x_k)$, by minimizing $||(x_1 - \widetilde{x_1}, \ldots, x_k - \widetilde{x_k})||_1 = \sum_i |x_i - \widetilde{x_i}|$, and $\widetilde{x_i} \in \text{domain}(O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}})$.

Incrementing Value (IV). For $(\widetilde{x_1}, \ldots, \widetilde{x_k})$, increment $O_\Delta^{\widetilde{T}}(\widetilde{x_1}, \ldots, \widetilde{x_k})$ by w, where w is the *largest* integer satisfying following *IV Constraints*:

C1. $w \leq \lceil O_\Delta^T(x_1, \ldots, x_k) \times s_T \rceil$, so the number of tuples for $(x_1, \ldots, x_k)$ is scaled by s_T.

Algorithm 2: Joint Degree Correlation

```
initialize O_Δ^T̃ as 0
for O_Δ^T(x_1, ..., x_k) and Σ_α̃ O_Δ^T̃(α̃) < |T̃| do
    (x̃_1, ..., x̃_k) ← 1-Norm(x_1, ..., x_k)
    w ← IV Calculation(x̃_1, ..., x̃_k)
    Value Update(w)
return O_Δ^T̃
```

C2. $w \leq \min_{1\leq i\leq k} O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}}(\widetilde{x_i})$, since the w tuples are constructed from tuples in $O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}}(\widetilde{x_i})$.

Value Update (VU). The w from the above ***IV*** calculation is used to update $O_\Delta^{\widetilde{T}}$ and $O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}}$, as follows:

(i) $O_\Delta^{\widetilde{T}}(\widetilde{x_1}, \ldots, \widetilde{x_k}) \leftarrow O_\Delta^{\widetilde{T}}(\widetilde{x_1}, \ldots, \widetilde{x_k}) + w$

(ii) $\forall i, O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}}(\widetilde{x_i}) \leftarrow O_\delta^{\widetilde{T_i}\rightarrow\widetilde{T}}(\widetilde{x_i}) - w$

We demonstrate these steps with $O_\Delta^A(2,2)$: Figure 7 shows $\text{domain}(O_\delta^{\widetilde{B}\rightarrow\widetilde{A}}) = \{1,2\}$, and $\text{domain}(O_\delta^{\widetilde{C}\rightarrow\widetilde{A}}) = \{2\}$, so for ***1-Norm Minimization***, the closest $(\widetilde{x_1}, \widetilde{x_2})$ is $(2,2)$. Figure 4 shows $O_\Delta^A(2,2) = 2$, so *C1* in ***IV*** calculation requires $w \leq \lceil 2 \times s_A \rceil = 4$; Figure 7 shows $O_\delta^{\widetilde{B}\rightarrow\widetilde{A}}(2) = 3$ and $O_\delta^{\widetilde{C}\rightarrow\widetilde{A}}(2) = 4$, so *C2* requires $w \leq \min\{3,4\} = 3$. Thus, $w = 3$. For ***VU***, $O_\Delta^{\widetilde{A}}(2,2)$ is incremented by 3, while $O_\delta^{\widetilde{B}\rightarrow\widetilde{A}}(2)$ and $O_\delta^{\widetilde{C}\rightarrow\widetilde{A}}(2)$ are each decremented by 3.

Figure 8 shows the resulting joint-degree distributions for $\widetilde{\mathcal{D}_{eg}}$ when Algorithm 2 terminates.

4.3 Step1c: Referencing Vector Correlation (RVC)

The next step towards $CoDa_{\widetilde{\mathcal{D}_{eg}}}$ construction in Figure 2 is Referencing Vector Correlation (RVC). JDC in the previous section correlates scalars (degrees) $\delta_{\widetilde{T_i}\rightarrow\widetilde{T}}$ to synthesize the vectors $\Delta_{\widetilde{T}}(\widetilde{t})$ in each FK of Figure 6. Recall from Definition 5 that the referencing vector $\phi_T(t)$ is the projection of $CoDa_{\mathcal{D}}$ on its FKs, so it is a vector of $\Delta_T(t)$ vectors. RVC, described below, correlates these $\Delta_{\widetilde{T}}(\widetilde{t})$ vectors for multiple FKs.

The algorithm for RVC is similar to JDC construction, except if T does not allow repeated FK tuples. For example, if T says user `uid` posted video `vid`, (`uid`, `vid`) should not appear twice in T.

DEFINITION 7. *For a table T, the FK matrix $\mathbb{F}_T$ is **unique** if and only if it does not have any repeated rows. Moreover, T is unique if and only $\mathbb{F}_T$ is unique.*

For example, C in Figure 3 is unique, but B is not. This property enforces an extra *uniqueness constraint* in addition to $C1$ and $C2$ presented in Section 4.2.

THEOREM 1. *Suppose table T references tables $T_1, \ldots, T_k$. If T is unique, then T satisfies the **uniqueness constraint**:*

$$O_\phi^T(\alpha_1, \ldots, \alpha_k) \leq \prod_{1\leq i\leq k} O_\Delta^{T_i}(\alpha_i). \tag{2}$$

Conversely, if this constraint is satisfied, then there is a unique T. □

The necessity in Theorem 1 says the number of tuples with RV $(\alpha_1, \ldots, \alpha_k)$ is bounded by the maximum number

$\widetilde{B}$
$O^{\widetilde{B}}_{\phi}(1,2)=1$
$O^{\widetilde{B}}_{\phi}(2,2)=6$

$\widetilde{C}$
$O^{\widetilde{C}}_{\phi}((2,2),(2))=4,\ O^{\widetilde{C}}_{\phi}((2,2),(1))=2$
$O^{\widetilde{C}}_{\phi}((1,2),(1))=2$

Figure 9: Referencing Vector Frequency Distribution for $\widetilde{\mathcal{D}_{eg}}$

Primary Key Vector	
$\widetilde{A}$	$\mathbb{P}_{\widetilde{A}}(1,2)=[\widetilde{a_1}]^\intercal\ \ \mathbb{P}_{\widetilde{A}}(2,2)=[\widetilde{a_2},\widetilde{a_3},\widetilde{a_4}]^\intercal$
$\widetilde{B}$	$\mathbb{P}_{\widetilde{B}}(2)=[\widetilde{b_1},\widetilde{b_2}]^\intercal, \mathbb{P}_{\widetilde{B}}(1)=[\widetilde{b_3},\widetilde{b_4},\widetilde{b_5},\widetilde{b_6}]^\intercal, \mathbb{P}_{\widetilde{B}}(0)=[\widetilde{b_7}]^\intercal$
$\widetilde{C}$	$\mathbb{P}_{\widetilde{C}}(0)=[\widetilde{c_1},\widetilde{c_2},\ldots,\widetilde{c_8}]^\intercal$

Figure 10: Primary Key Vector for $\widetilde{\mathcal{D}_{eg}}$

of possibilities. Section 5.2.2 will address the sufficiency of this constraint. RVC scales O^T_ϕ to $O^{\widetilde{T}}_\phi$ like in JDC:

1-Norm Minimization. For $(\alpha_1,\ldots,\alpha_k)$, choose the **closest** $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$ where $\sum_{1\le i\le k}||\alpha_i-\widetilde{\alpha_i}||_1$ is minimized, and $\widetilde{\alpha_i}\in \text{domain}(O^{\widetilde{T_i}}_\Delta), \forall 1\le i\le k$.

Incrementing Value (IV). For each $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$, increment $O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$ by w, where w is the *largest* integer satisfying the *IV Constraints*:

C1'. $w\le\lceil O^T_\phi(\alpha_1,\ldots,\alpha_k)\times s_T\rceil$. Similar to JDC, C1' enforces the incremental value proportional to $O^T_\phi(\alpha_1,\ldots,\alpha_k)$.

C2'. $w\le\min_{1\le i\le k}\Upsilon_{\widetilde{T_i}\to\widetilde{T}}(\widetilde{\alpha_i})$. Since $O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$ is based on $O^{\widetilde{T_1}}_\Delta(\widetilde{\alpha_1}),\ldots,O^{\widetilde{T_k}}_\Delta(\widetilde{\alpha_k})$, the maximum number of tuples incremented should not be larger than the maximum appearance allowed for each $\Upsilon_{\widetilde{T_i}\to\widetilde{T}}(\widetilde{\alpha_i})$.

C3'. If T is unique, then $w\le\prod_{1\le i\le k}O^{\widetilde{T_i}}_\Delta(\widetilde{\alpha_i})-O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$, as required by Theorem 1.

Value Update (VU). After w is calculated, the corresponding distributions are updated, as follows:

(i) $O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})\leftarrow O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})+w$

(ii) $\forall i, \Upsilon_{\widetilde{T_i}\to\widetilde{T}}(\widetilde{\alpha_i})\leftarrow\Upsilon_{\widetilde{T_i}\to\widetilde{T}}(\widetilde{\alpha_i})-w$

We demonstrate the above for $O^C_\phi((2,2),(2))=2$ in Figure 4:

For ***1-Norm Minimization***, $\text{domain}(O^{\widetilde{A}}_\Delta)=\{(1,2),(2,2)\}$, and $\text{domain}(O^{\widetilde{B}}_\Delta)=\{(0),(1),(2)\}$ in Figure 8. Thus, the closest $(\widetilde{\alpha_1},\widetilde{\alpha_2})$ is $((2,2),(2))$.

For ***IV***, by *C1'*, $w\le\lceil 2\times s_C\rceil=4$. By *C2'*, $w\le\min\{\Upsilon_{\widetilde{C}\to\widetilde{A}}(2,2),\Upsilon_{\widetilde{C}\to\widetilde{B}}(2)\}=\min\{6,4\}$ by Figure 6. Since C is unique, *C3'* requires $w\le O^{\widetilde{A}}_\Delta(2,2)\times O^{\widetilde{B}}_\Delta(2)$ $-O^{\widetilde{C}}_\phi((2,2),(2))=3\times 2-0$, by Figure 8. Hence, $w=4$.

For ***VU***, (i) $O^{\widetilde{C}}_\phi((2,2),(2))\leftarrow 0+4$, (ii) $\Upsilon_{\widetilde{C}\to\widetilde{A}}(2,2)\leftarrow 6-4$, and $\Upsilon_{\widetilde{C}\to\widetilde{B}}(2)\leftarrow 4-4$.

When the RVC loop terminates, we get the scaled referencing vector frequency distribution for $\widetilde{\mathcal{D}_{eg}}$ in Figure 9.

Due to the uniqueness constraint, it is possible that

$$G_{\widetilde{T}}=\sum_{\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k}}O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})<|\widetilde{T}| \quad (3)$$

To increase $G_{\widetilde{T}}$, we can generate some random tuples. Specifically, $G_{\widetilde{T}}<|\widetilde{T}|$ implies there are some $\Upsilon_{\widetilde{T_1}\to\widetilde{T}}(\widetilde{\theta_1})>0,\ldots,\Upsilon_{\widetilde{T_k}\to\widetilde{T}}(\widetilde{\theta_k})>0$, so DSCALER randomly combines them to form new tuples $\widetilde{t}$ with $\phi_{\widetilde{T}}(\widetilde{t})=(\widetilde{\theta_1},\ldots,\widetilde{\theta_k})$, without violating the uniqueness constraint. If $G_{\widetilde{T}}<|\widetilde{T}|$ is still true, we choose $\widetilde{\theta_1},\ \ldots,\ \widetilde{\theta_k}$ from $\Upsilon_{\widetilde{T_1}\to\widetilde{T}},\ldots,\Upsilon_{\widetilde{T_k}\to\widetilde{T}}$ to **swap** with some generated $O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$. The idea is to remove 1 generated tuple with referencing vector $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$, and form two new tuples, $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_{i-1}},\widetilde{\theta_i},\widetilde{\alpha_{i+1}},\ldots,\widetilde{\alpha_k})$ and $(\widetilde{\theta_1},\ldots,\widetilde{\theta_{i-1}},\widetilde{\alpha_i},\widetilde{\theta_{i+1}},\ldots,\widetilde{\theta_k})$. The net effect of one successful swap increases $O^{\widetilde{T}}_\phi$ by 1 tuple.

In the worst case, $G_{\widetilde{T}}<|\widetilde{T}|$ remains true after this swapping. We then generate $|\widetilde{T}|-G_{\widetilde{T}}$ ***shadow tuples*** (see Section 5.4).

4.4 Step1d: Key Vector Correlation(KVC)

The last step in Figure 2 for constructing $CoDa_{\widetilde{\mathcal{D}}}$ from $CoDa_{\mathcal{D}}$ is Key Vector Correlation (KVC). This construction to correlate primary and foreign keys is similar to the above RVC for correlating FKs, so we omit the details here; the reader can refer to the pseudo-code in the full paper[31]. After KVC, the scaled $CoDa_{\widetilde{\mathcal{D}_{eg}}}$ is as shown in Figure 6.

5. PHASE2: VALUE SYNTHESIS

The tuples in $CoDa_{\widetilde{\mathcal{D}}}$ generated by Phase1 only contain degree correlation patterns. Phase2 must now replace these correlation patterns with synthetic attribute values, and add non-key values. The 4 steps in Figure 2 are described below.

5.1 Step2a: PK Vector Synthesis

For joint degree $\widetilde{\alpha}$ and $O^{\widetilde{T}}_\Delta(\widetilde{\alpha})=w$, Step2a must generate w unique PKs to give a vector $\mathbb{P}_{\widetilde{T}}(\widetilde{\alpha})$. For $\widetilde{\mathcal{D}_{eg}}$, Figure 8 shows $O^{\widetilde{A}}_\Delta(1,2)=1$ and $O^{\widetilde{A}}_\Delta(2,2)=3$, so Step2a generates one tuple with PK $\widetilde{a_1}$ and joint degree $(1,2)$, and 3 tuples with PKs $\widetilde{a_2},\widetilde{a_3},\widetilde{a_4}$ and joint degree $(2,2)$. Figure 10 shows the PK vectors $\widetilde{\mathcal{D}_{eg}}$.

5.2 Step2b: FK Matrix Synthesis

Next Step2b generates the FK matrix $\mathbb{F}_{\widetilde{T}}$, where $\widetilde{T}$ references $\widetilde{T_1},\ldots,\widetilde{T_k}$. For $O^{\widetilde{T}}_\phi(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})=w$, tuples $\widetilde{t_1},\ldots,\widetilde{t_w}$ are generated for $\mathbb{F}_{\widetilde{T}}$, each with RV $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$. We first sketch the basic ideas and main challenges, before describing the DSCALER solution.

5.2.1 Basic ideas: column and row generation

To generate the submatrix of $\mathbb{F}_{\widetilde{T}}$ for RV $(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$, we can proceed in two steps:

Column Generation (CG). For the k columns $\widetilde{\mathbb{C}_1},\ldots,\widetilde{\mathbb{C}_k}$ of $\mathbb{F}_{\widetilde{T}}(\widetilde{\alpha_1},\ldots,\widetilde{\alpha_k})$, select w PK elements from $\widetilde{T_i}$ for each $\widetilde{\mathbb{C}_i}$, such that the joint degree $\Delta_{\widetilde{T_i}}(\widetilde{e})=\widetilde{\alpha_i}, \forall\widetilde{e}\in\widetilde{\mathbb{C}_i}$. However, there are many possibilities for choosing these w elements. In $\widetilde{\mathcal{D}_{eg}}$, Figure 9 shows $O^{\widetilde{C}}_\phi((2,2),(1))=2$, so we need to choose the 2 elements for FK_A with joint degree $(2,2)$. Figure 10 shows $\mathbb{P}_{\widetilde{A}}(2,2)=[\widetilde{a_2},\widetilde{a_3},\widetilde{a_4}]^\intercal$, so there are 3 candidates, and each can appear 2 times. Thus, there are 6 possibilities for $\widetilde{\mathbb{C}_1}$: $\{\widetilde{a_2},\widetilde{a_2}\}$, $\{\widetilde{a_3},\widetilde{a_3}\}$, $\{\widetilde{a_4},\widetilde{a_4}\}$, $\{\widetilde{a_2},\widetilde{a_3}\}$, $\{\widetilde{a_2},\widetilde{a_4}\}$, $\{\widetilde{a_3},\widetilde{a_4}\}$.

Row Generation (RG). After CG generates the column elements, we need to generate the rows by matching elements in different columns. This matching cannot be arbitrary if $\widetilde{T}$ has the uniqueness constraint. For example,

$$\widetilde{\mathbb{C}_1} = \begin{bmatrix} a \\ a \\ b \\ b \end{bmatrix} \quad \widetilde{\mathbb{C}_2} = \begin{bmatrix} d \\ d \\ e \\ e \end{bmatrix} \quad M_1 = \begin{bmatrix} a & d \\ a & d \\ b & e \\ b & e \end{bmatrix} \quad M_2 = \begin{bmatrix} a & d \\ a & e \\ b & d \\ b & e \end{bmatrix}$$

Figure 11: Row Generation (RG): M_1 violates uniqueness.

in Figure 11, the permutation in M_1 yields repeated rows, whereas that in M_2 has no repeated rows.

There are two challenges in column and row generation: (i) The choice of elements in column generation should facilitate enforcement of the uniqueness constraint. For example, if $\widetilde{\mathbb{C}_1} = [a, a, a, b]^\intercal$ in Figure 11, then there is no way of getting unique rows by permuting the columns. (ii) If there is a permutation of the column elements that can generate unique rows, then it must be found with minimal effort.

5.2.2 Uniqueness from uniformity

Intuitively, to enforce the uniqueness constraint is to maximize the variety in the rows. We can achieve this by ensuring that the column elements have a uniform number of repetitions, as follows:

Uniform Column Generation (uCG). For each $\mathbb{F}_{\widetilde{T}}(\widetilde{\alpha_1}, \ldots, \widetilde{\alpha_k}) = w$, suppose there are n_i tuples in $\widetilde{T_i}$ with joint degree $\widetilde{\alpha_i}$, and they appear as FK in $\widetilde{T}$ for w times; uCG distributes these w appearances evenly to the n_i tuples, so each element appears $\lceil \frac{w}{n_i} \rceil$ or $\lfloor \frac{w}{n_i} \rfloor$ times in $\widetilde{\mathbb{C}_i}$. Thus, for each $\widetilde{\mathbb{C}_i}$, its elements' appearances differ by at most 1. This allows maximum variety and facilitates uniqueness in row generation below.

Unique Row Generation (uRG). If the desired number of rows w exceeds the possible number of unique rows, then (by Pigeonhole Principle) some rows of $\mathbb{F}_{\widetilde{T}}$ must repeat; otherwise, we can get w unique rows by a straightforward lexicographic ordering. This intuition says the inequality in Theorem 1 is sufficient for uniqueness, and is confirmed by the theorems below. (Space constraint prevents us from presenting the proofs, which can be found in the full paper[31].) In the following, u^m in a multiset denotes m copies of u.

THEOREM 2. *Given multiset* $\mathbb{U} = \{u_1^{m_{u_1}}, \ldots, u_{n_1}^{m_{u_{n_1}}}\}$, $\mathbb{V} = \{v_1^{m_{v_1}}, \ldots, v_{n_2}^{m_{v_{n_2}}}\}$, *and*

$$\sum_{u_i \in \mathbb{U}} m_{u_i} = |\mathbb{U}| = |\mathbb{V}| = \sum_{v_j \in \mathbb{V}} m_{v_j}$$

$$\max_{u_i \in \mathbb{U}} m_{u_i} - \min_{u_i \in \mathbb{U}} m_{u_i} \leq 1$$

$$\max_{v_j \in \mathbb{V}} m_{v_j} - \min_{v_j \in \mathbb{V}} m_{v_j} \leq 1.$$

Let $w = \langle w(0), w(1) \rangle$ *where* $w(0) \in \mathbb{U}$ *and* $w(1) \in \mathbb{V}$ *denote a* **match**. *Then there exists a multiset of matches* $\mathbb{W} = \{w_1, w_2, \ldots\}$ *such that*

$$\mathbb{U} = \bigcup_k \{w_k(0)\}, \ \mathbb{V} = \bigcup_k \{w_k(1)\}$$

$$\max_{w_k \in \mathbb{W}} m_{w_k} - \min_{w_k \in \mathbb{W}} m_{w_k} \leq 1$$

$$\forall w_k \in \mathbb{W}, m_{w_k} \leq \lceil \frac{|\mathbb{W}|}{n_1 \times n_2} \rceil.$$

□

Table	Foreign Key Matrix
$\widetilde{B}$	$\mathbb{F}_{\widetilde{B}}(2,2) = [[\widetilde{a_2}], [\widetilde{a_3}], [\widetilde{a_2}], [\widetilde{a_3}], [\widetilde{a_4}], [\widetilde{a_4}]]^\intercal$, $\mathbb{F}_{\widetilde{B}}(1,2) = [[\widetilde{a_1}]]^\intercal$
$\widetilde{C}$	$\mathbb{F}_{\widetilde{C}}((2,2),(2)) = [[\widetilde{a_2}, \widetilde{b_1}], [\widetilde{a_2}, \widetilde{b_2}], [\widetilde{a_3}, \widetilde{b_1}], [\widetilde{a_4}, \widetilde{b_2}]]^\intercal$ $\mathbb{F}_{\widetilde{C}}((2,2),(1)) = [[\widetilde{a_3}, \widetilde{b_3}], [\widetilde{a_4}, \widetilde{b_4}]]^\intercal$ $\mathbb{F}_{\widetilde{C}}((1,2),(1)) = [[\widetilde{a_1}, \widetilde{b_5}], [\widetilde{a_1}, \widetilde{b_6}]]^\intercal$

Figure 12: Foreign Key Matrix for $\widetilde{\mathcal{D}_{eg}}$

Theorem 2 says that, suppose we have two multisets $\mathbb{U}, \mathbb{V}$ with the same cardinality, each has n_1, n_2 different elements respectively, and the maximum and minimum multiplicity in each multiset differ by at most 1. If we match up all the elements in $\mathbb{U}, \mathbb{V}$ to form a matching set $\mathbb{W}$, then there is a matching set $\mathbb{W}$ where the maximum and minimum multiplicity of the pairings differ by at most 1 as well. Moreover, the maximum multiplicity is not greater than $\lceil \frac{|\mathbb{W}|}{n_1 \times n_2} \rceil$. It follows that, if $|\mathbb{W}| \leq n_1 n_2$, then the multiplicities are at most 1, so the matches are unique.

We can generalize this result to k multisets:

THEOREM 3. *For* k *multisets* $\mathbb{U}_1, \ldots, \mathbb{U}_k$ *with equal cardinality,* n_i *is the number of different elements in* $\mathbb{U}_i$ *and*

$$\forall \mathbb{U}_i, \max_{e_j \in \mathbb{U}_i} m_{e_j} - \min_{e_j \in \mathbb{U}_i} m_{e_j} \leq 1,$$

then there is a matching multiset $\mathbb{W}$ *with maximum multiplicity* $\lceil \frac{|\mathbb{W}|}{\prod_{1 \leq i \leq k} n_i} \rceil$. □

In the DSCALER context, $\mathbb{U}_i$ are the elements of $\mathbb{C}_i$, each match is a row of $\mathbb{F}_{\widetilde{T}}$, and lexicographic ordering generates a unique $\mathbb{F}_{\widetilde{T}}$ if and only if $|\mathbb{W}| \leq \prod_{1 \leq i \leq k} n_i$. We thus get:

THEOREM 4. DSCALER*'s uCG and uRG generate a unique* $\mathbb{F}_{\widetilde{T}}(\widetilde{\alpha_1}, \ldots, \widetilde{\alpha_k})$ *if and only if the inequality in Theorem 1 is satisfied.* □

For $\widetilde{\mathcal{D}_{eg}}$ in Figure 3, $\widetilde{C}$ is unique, but $\widetilde{B}$ is not. We demonstrate $\mathbb{F}_{\widetilde{T}}$ generation for the case $O_\phi^{\widetilde{C}}((2,2),(2)) = 4$ from Figure 9: By Figure 10, $\mathbb{P}_{\widetilde{A}}(2,2)$ shows $\widetilde{\mathbb{C}_1}$ has elements $\widetilde{a_2}$, $\widetilde{a_3}$ and $\widetilde{a_4}$, while $\mathbb{P}_{\widetilde{B}}(2)$ shows $\widetilde{\mathbb{C}_2}$ has elements $\widetilde{b_1}$ and $\widetilde{b_2}$. The desired number of tuples is $w = 4$, so by uCG, we get $\widetilde{\mathbb{C}_1} = \{\widetilde{a_2}, \widetilde{a_2}, \widetilde{a_3}, \widetilde{a_4}\}$ and $\widetilde{\mathbb{C}_2} = \{\widetilde{b_1}, \widetilde{b_1}, \widetilde{b_2}, \widetilde{b_2}\}$. Lexicographic ordering in uRG then generates 4 unique matches $[\widetilde{a_2}, \widetilde{b_1}]$, $[\widetilde{a_2}, \widetilde{b_2}]$, $[\widetilde{a_3}, \widetilde{b_1}]$ and $[\widetilde{a_4}, \widetilde{b_2}]$, as desired.

Figure 12 shows the FK matrices generated by uCG and uRG for all the tables in $\widetilde{\mathcal{D}_{eg}}$.

5.3 Step2c: Key Matrix Synthesis

So far, Step2a in Section 5.1 has generated the PK vector $\mathbb{P}_{\widetilde{T}}$ and Step2b in Section 5.2 has generated the FK matrix $\mathbb{F}_{\widetilde{T}}$ for each $\widetilde{T}$. Step2c now uses the KV distribution $O_\Phi^{\widetilde{T}}$ to match the PKs and FKs to generate the key matrix $\mathbb{K}_{\widetilde{T}}$.

For $O_\Phi^{\widetilde{T}}(\widetilde{\alpha}, \widetilde{\beta}) = w$, we generate $\mathbb{K}_{\widetilde{T}}(\widetilde{\alpha}, \widetilde{\beta})$ by pairing w randomly chosen elements from $\mathbb{P}_{\widetilde{T}}(\widetilde{\alpha})$ and w randomly chosen elements from $\mathbb{F}_{\widetilde{T}}(\widetilde{\beta})$.

We demonstrate this pairing for $O_\Phi^{\widetilde{B}}((2,2),(2)) = 2$ in Figure 6. Figure 10 shows $\mathbb{P}_{\widetilde{B}}(2) = [\widetilde{b_1}, \widetilde{b_2}]^\intercal$ and Figure 12 shows $\mathbb{F}_{\widetilde{B}}(2,2) = [[\widetilde{a_2}], [\widetilde{a_3}], [\widetilde{a_2}], [\widetilde{a_3}], [\widetilde{a_4}], [\widetilde{a_4}]]^\intercal$. To generate $w = 2$ tuples, we use $\widetilde{b_1}$, $\widetilde{b_2}$ from $\mathbb{P}_{\widetilde{B}}(2)$ and (randomly) choose $\widetilde{a_2}$, $\widetilde{a_3}$ from $\mathbb{F}_{\widetilde{B}}(2,2)$ to form $[\widetilde{b_1}, \widetilde{a_2}]$, $[\widetilde{b_2}, \widetilde{a_3}]$ for submatrix $\mathbb{K}_{\widetilde{B}}((2),(2,2))$. Figure 5 presents the scaled key matrices.

5.4 Step2d: Non-Key Value Synthesis

Phase1 in Figure 2 omits the non-key attributes in scaling $CoDa_{\widetilde{\mathcal{D}}}$. Phase2 therefore needs a last Step2d to reintroduce non-key attributes. For this paper, we reuse non-key values from $\mathcal{D}$, instead of generating synthetic values.

To see the underlying issue, consider a table `book-review` where each review has two non-key values d and n, for the date and number of "likes". Reproducing the correlation between these two values will require assumptions about their domains and joint distributions. Space constraint prevents us from examining this issue. For now, we would simply reproduce the correlation by sampling (d, n) pairs from $\mathcal{D}$.

In general, DSCALER assigns the non-key values based on the tuples' key vector KV. For all the non-key values, DSCALER assigns the non-key values based on the closest KV. For example, $\widetilde{t}$ in scaled table $\widetilde{T}$ has KV $\widetilde{\gamma}$. In the original table T, the KV that is closest (see Section 4.2) to $\widetilde{\gamma}$ is γ, and $t_1, \ldots, t_k$ in T are the tuples having KV γ. DSCALER randomly selects non-key values from $t_1, \ldots, t_k$ to assign to $\widetilde{t}$. Since DSCALER treats the non-key values as a whole, DSCALER thus maintains the non-key correlations better than non-key attribute assignment.

As pointed out at the end of Section 4.3, there may be a few *shadow tuples* resulting from *RVC*. For these shadow tuples, we just randomly fill in the values without violating the uniqueness constraint. In our experiment, the number of shadow tuples is small ($< 1\%$ for our experiments, on average), and happens for one data set only (`DoubanBook`).

6. EXPERIMENTAL EVALUATION

This section describes the alternative algorithms, datasets and similarity measures used in our experiments.

State-of-the-Art Alternatives

As explained in Section 2, we focus our DSCALER comparison on two state-of-the-art algorithms:

1. VFDS is a random sampling approach for scaling *down* a relational database [8]. VFDS first chooses a most relevant starting table T_* from $\mathcal{D}$, then randomly selects the tuples to form $O(T_*)$. Next, VFDS enlarges $O(T_*)$ by sampling tuples associated with $O(T_*)$ (references or being referenced). It stops sampling when all tables have been visited. VFDS thus samples a subset of the original dataset by following foreign keys. In contrast, DSCALER constructs the required number of correlation patterns (Phase1), then replace those patterns with key values (Phase2). Buda et al.'s experiments show that VFDS is about 300 times faster than previous sampling approaches, and provides similar query answers.

2. UpSizeR is the first solution to DSP for a scalar scaling factor s [29]. It generates tables in the (partial) order dictated by the FK references, starting with tables that have no foreign keys. For a table with just one FK, this generation is based on the degree distribution; for a table with multiple FKs, UpSizeR clusters the values in each FK, does co-clustering[10], then generates FK values per co-cluster. In contrast, DSCALER replicates per-tuple correlation patterns, thus facilitating non-uniform scaling and greater similarity. UpSizeR does not guarantee uniqueness (Definition 7).

Datasets

We present experiments on 3 datasets, 2 from previous work and 1 newly crawled (schemas in the full paper[31]).

1. `TPC-H`[2] is a synthetic database containing typical business data, such as suppliers and orders; there are 8 tables with a well-known schema. It is generated by `dbgen` and the size is set to 1GB. This dataset was used by UpSizeR.

2. `financial` database[3] first appeared in PKDD'99 Challenge Discovery. It contains typical financial data, such as loans and transactions. `financial` has 8 tables, and the table size (number of tuples) ranges from 77 (District) to 1056320 (Transactions). This dataset was used by VFDS.

3. `DoubanBook`[4] was crawled from `book.douban.com`, a Chinese social network website that allows the creation and sharing of content related to movies, books, music, and recent events and activities in Chinese cities. `DoubanBook` contains the book-related data. It has 12 tables and 39082997 tuples, and the table size ranges from 686605 to 12891598. `DoubanBook` is different from the previous two datasets by having two heavily referenced table, `user` and `book`. Such a property is very common in social network datasets.

Similarity Measure

The similarity of the scaled database and original database is measured by the following properties:

Table Size Error. The relative error of the table size is $e_{\widetilde{T}} = \frac{||\widetilde{T}| - s_T|T||}{s_T|T|}$. For a database $\widetilde{\mathcal{D}} = \{\widetilde{T_1}, \ldots, \widetilde{T_K}\}$, the relative error of database size is $e_{\widetilde{\mathcal{D}}} = (\sum_i e_{\widetilde{T_i}})/K$.

Query Error. The database semantic similarity is measured by the result of an aggregate query. $q(\mathcal{D}')$ is the (numerical) result when query q runs on database $\mathcal{D}'$. For *uniform* scaling factor s, the relative error for a count query q is $e_{\texttt{COUNT}} = \frac{|q(\widetilde{\mathcal{D}}) - s \times q(\mathcal{D})|}{s \times q(\mathcal{D})}$, while the relative error for an average query q is $e_{\texttt{AVE}} = \frac{|q(\widetilde{\mathcal{D}}) - q(\mathcal{D})|}{q(\mathcal{D})}$.

Distribution Error. Queries are application-specific: different datasets will not have the same set of queries, especially if the datasets are from different domains. Hence we also need a more generic (application-independent) measure for similarity. Here, we use the frequency distribution for degree as defined in Section 3. To measure the similarity of two probability distributions, we use the Kolmogorov-Smirnov (KS) D-statistic $\sup_x |F_1(x) - F_2(x)|$, where F_1 and F_2 are cumulative distribution functions (cdf). For a database $\widetilde{D}$, we take the average KS D-statistic for all FK references.

Results Summary

All experiments are done on a Linux machine with 128GB memory and AMD Opteron 2.3GHz processor. For UpSizeR, we use the original C++ implementation[4]. For VFDS, we also use the original Java implementation[5]. DSCALER is implemented in Java.

The experiments show that DSCALER is consistently one of the best-performing algorithms. VFDS sometimes matches DSCALER's performance for `TPC-H` and `financial`, but not for `DoubanBook` (e.g. Figure 13). Similarly, UpSizeR sometimes matches DSCALER's performance, but has unreasonably long execution time for `DoubanBook` (e.g. Figure 16). In fact, *if a plot does not include* UpSizeR, *that indicates it takes too long to finish.* Section 7 and Section 8 present details for *uniform* and *nonuniform* scaling respectively.

[2] https://www.tpc.org/tpch/
[3] http://lisp.vse.cz/pkdd99/berka.htm
[4] http://www.comp.nus.edu.sg/~upsizer/
[5] https://github.com/tbuda

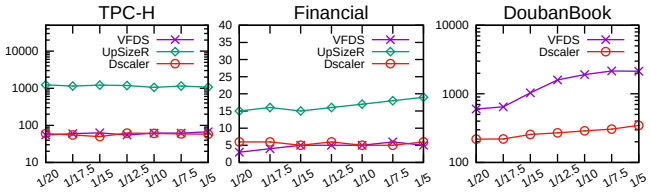

Figure 13: Execution Time (seconds, log scale) for Scaling Down.

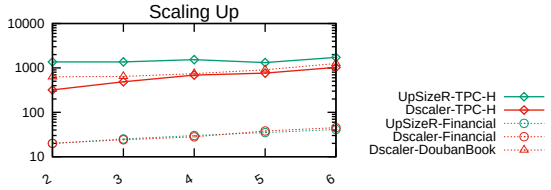

Figure 14: Execution Time (seconds, log scale) for Scaling Up

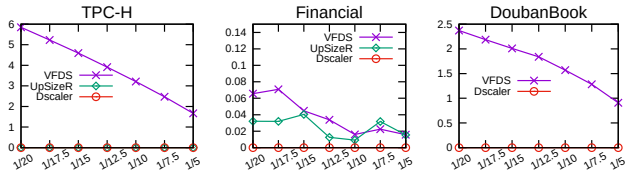

Figure 15: Table Size Error for Scaling Down

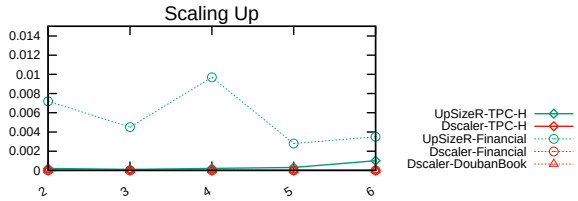

Figure 16: Table Size Error for Scaling Up.

7. UNIFORM SCALING

Previous algorithms *all* use a *single* scaling factor s, so we do the same here with $s = \frac{1}{5}, \frac{1}{7.5}, \frac{1}{10}, \frac{1}{12.5}, \frac{1}{15}, \frac{1}{17.5}, \frac{1}{20}$ for scaling down, and $s = 2, 3, 4, 5, 6$ for scaling up. In the plots, the horizontal axis is s, *purple* color is for VFDS, *green* is for UpSizeR, and DSCALER is *red*.

Execution Time

Figure 13 presents the results for scaling down. For `TPC-H` and `financial`, DSCALER and VFDS are equally efficient. For `DoubanBook`, VFDS is much slower (note the log scale) than DSCALER, and UpSizeR takes unreasonably long to finish. `DoubanBook` has two tables, `user` and `book`, that are heavily referenced by others; they dramatically increase the run time for VFDS and UpSizeR.

For scaling up, VFDS does not apply, so Figure 14 only compares UpSizeR and DSCALER. It shows UpSizeR is comparable to DSCALER for the small `financial` dataset, slower for `TPC-H` and, again, unreasonably slow for `DoubanBook`.

Table Size Error

For scaling down, Figure 15 shows that DSCALER has negligible Table Size Error (see Tuple Scaling in Step1a). UpSizeR is similar, but has noticeable errors for `financial`. The error for VFDS increases as the sample size gets smaller (contrary to the point of scaling down, where s is small).

For scaling up, Figure 16 shows that UpSizeR is comparable to DSCALER for `TPC-H` and has larger errors for `financial`; it does not finish in time for a comparison to DSCALER for `DoubanBook`.

Query Error

To measure query error for `TPC-H`, we adapt queries Q1–Q5 used for UpSizeR [29, 31]. For scaling down, Figure 17 shows UpSizeR and DSCALER have similar performance. VFDS does badly for Q3 and Q5, likely because of its large Table Size Errors (Figure 15). For scaling up, Figure 18 shows that both DSCALER and UpSizeR have small errors.

The queries for `financial` are categorized into four groups [7, 8]: (G1) queries that compute an aggregate value on **key** attributes; (G2) queries that compute an aggregate value on **non-key** attributes; (G3) queries that compute an aggregate value on **key** attributes with a **WHERE** clause; (G4) queries that compute an aggregate value on **non-key** attributes with a **WHERE** clause. We run 3 queries for each group, so there are 12 queries [31]. Due to space constraint, we present only the average error for each group. For scaling down, Figure 19 shows UpSizeR has errors that are similar to DSCALER for G1, G2 and G3, but generally larger for G4. The VFDS errors are similar to DSCALER's for G3 and G4, but larger for G1 and G2. For scaling up, Figure 20 shows that the UpSizeR errors are similar to DSCALER's for G1 and G3, but larger for G2 and G4.

For `DoubanBook`, we use queries Q1, ..., Q8 that are designed to be meaningful [31]; e.g. Q3 is "select the number of authors whose books have been read by some users". Since `DoubanBook` is too big for UpSizeR, we only compare VFDS and DSCALER for scaling down. Figure 21 shows that DSCALER has negligible errors for all 8 queries, whereas VFDS is accurate only for Q2 and Q6. For scaling up, there is no comparison since VFDS only scales down, and UpSizeR fails for `DoubanBook`; Figure 22 shows DSCALER has very small errors, except for Q3 (at most 0.06).

Distribution Error

For distribution error, Figure 23 shows that all 3 algorithms have similarly small errors for scaling down `financial`, but DSCALER is best for `TPC-H` and `DoubanBook`. For scaling up, Figure 24 shows negligible error for DSCALER; UpSizeR's errors for `TPC-H` and `financial` are also small (< 0.01).

8. NONUNIFORM SCALING

Since previous algorithms do not perform nonuniform scaling, this section only presents results for DSCALER.

Dataset Partitioning

For a dataset $\mathcal{D}$, we can think of its various sizes as determined by growth. Hence, let D_i be a snapshot of $\mathcal{D}$ at timestamp $i = 1, \ldots, n$, and assume $D_1 \subset D_2 \subset \cdots \subset D_n$. By taking the ratio of D_i and D_1's table sizes, we get the scaling factor $s_T (\geq 1)$ for each table T.

For `TPC-H`, we use the default `dbgen` tool to generate the datasets D_1, D_2, D_3, D_4, D_5; D_1 is the starting database with size $1GB$, and D_2, D_3, D_4, D_5 are the databases of size $3GB$, $5GB$, $7GB, 9GB$ respectively.

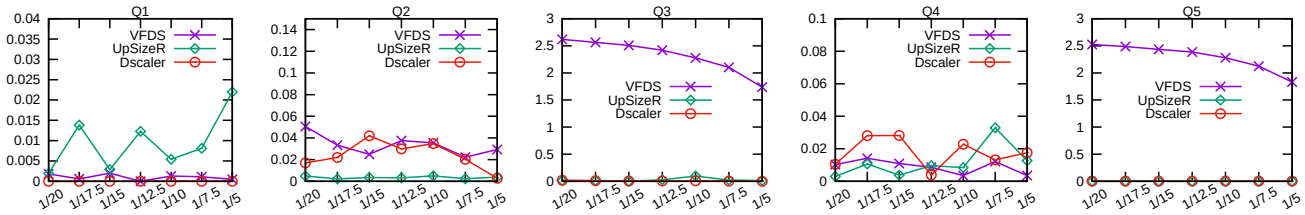

Figure 17: TPC-H Query Error for Scaling Down

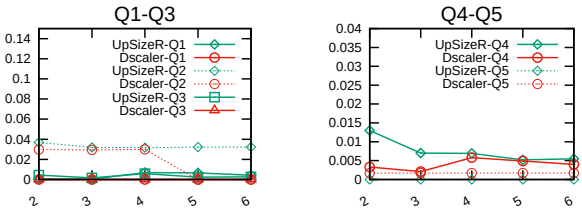

Figure 18: TPC-H Query Error for Scaling Up

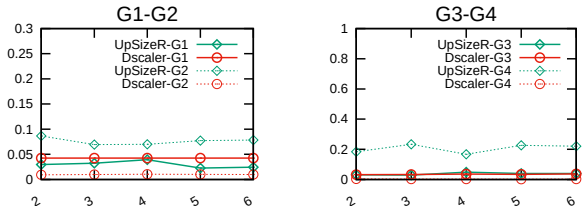

Figure 20: financial Query Error for Scaling Up

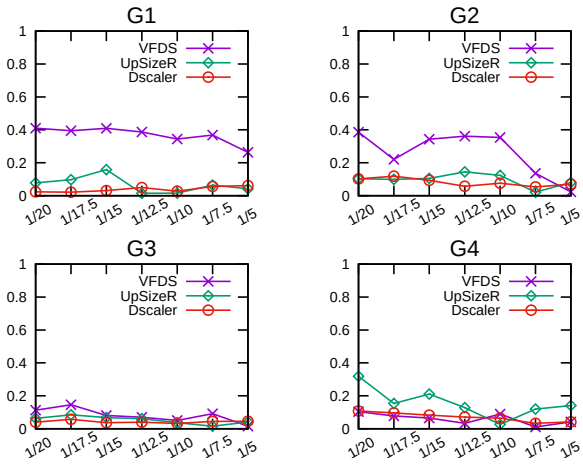

Figure 19: financial Query Error for Scaling Down

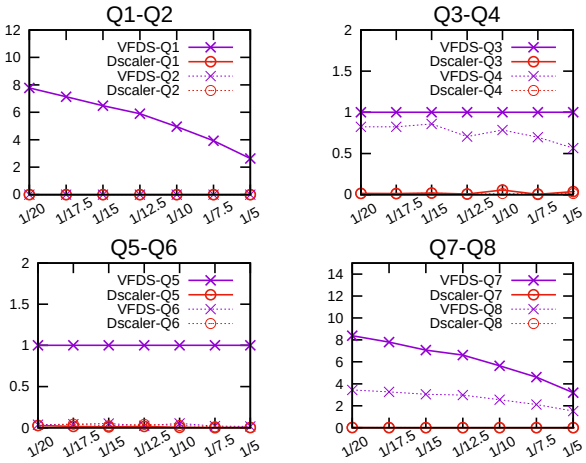

Figure 21: DoubanBook Query Error for Scaling Down

For financial, we grow it chronologically. D_1 contains all tuples (and associated tuples) with date < 1997-10-01, and D_2, D_3, D_4, D_5 contain all tuples with dates < 1998-01-01, 1998-04-01, 1998-07-01, 1998-10-01 respectively.

For DoubanBook, the dataset is obtained similarly, but with a longer time span. D_1 contains all tuples (and associated tuples) with date < 2012-07-01, and D_2, D_3, D_4, D_5 contain all tuples with dates < 2013-01-01, 2013-07-01, 2014-07-01, 2015-07-01 respectively.

With this partitioning, TPC-H grows almost uniformly, in contrast to financial and DoubanBook. For example, in DoubanBook's D_1, the user and book-wish table sizes are 2013879 and 4582596 but, in D_5, the user and book-wish table sizes are 3519093 and 12891598. Thus, book-wish and user scale by 3 and 1.7 times, respectively.

Since DSCALER's Step1a ensures near-zero table size errors, we only report Query and Distribution Errors below.

Query Error

For uniform scaling, Query Error is measured against the input $\mathcal{D}$, using s in the case of q_{COUNT}. For nonuniform scaling, the tables grow at different rates, so we measure Query Error by comparing the scaled $\widetilde{D_i}$ to the empirical D_i instead; i.e. for query q (COUNT or AVE), error $e_q = \frac{|q(\widetilde{D_i}) - q(D_i)|}{q(D_i)}$. We use the same queries from Section 7. The results for $\widetilde{D_2}$, $\widetilde{D_3}$, $\widetilde{D_4}$, and $\widetilde{D_5}$ are grouped together by query.

For TPC-H, Figure 25 shows about 0.07 error for Q2 on D_5, but much less (< 0.005) for other queries and D_i. For financial, the queries are more complex, but the errors are at most 0.12 (for G4). For DoubanBook, although the database is partitioned by longer time spans, DSCALER still produces $\widetilde{D_i}$ with less than 0.1 error for most queries, except Q1 (with 0.15 average error).

Distribution Error

Figure 26 shows that DSCALER has negligible distribution errors for TPC-H; this is expected, since dbgen scales the tables in TPC-H almost uniformly. The errors for financial are consistently small (≈ 0.005). For DoubanBook, the errors are larger, but still small (< 0.04).

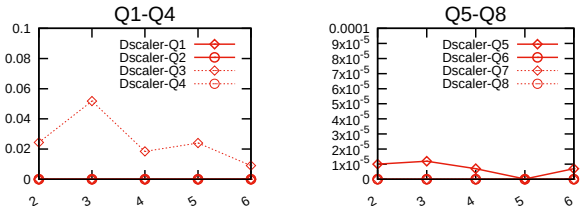

Figure 22: `DoubanBook` Query Error for Scaling Up

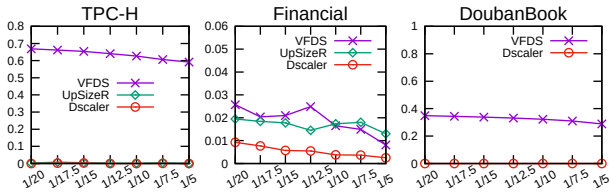

Figure 23: Distribution Error for Scaling Down

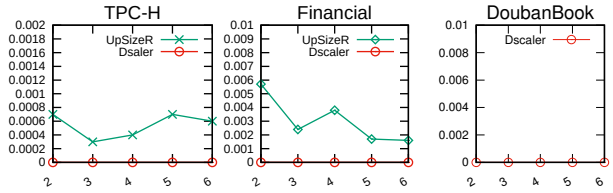

Figure 24: Distribution Error for Scaling Up

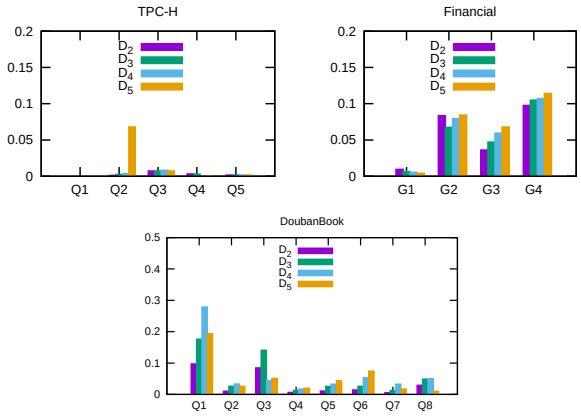

Figure 25: Query Error for Nonuniform Scaling

9. LIMITATIONS

DSCALER aims to reproduce features in $\mathcal{F}$, so (naturally) it may not do well for features not in $\mathcal{F}$. Due to space constraint, we list just two examples:

(1) Join Topologies

Consider `COUNT` queries on `DoubanBook`. For $n \geq 1$, define a **star** n**-join** to be a query that joins $n+1$ tables on the *same* key, and a **linear** n**-join** to be a query that joins $n+1$ tables on n *different* keys. Further, **star** 0**-join** and **linear** 0**-join** both refer to a selection on a single table (no join), such as "number of books which were read by some user".

For examples of a star topology, "number of books which have been read and commented by some users" is a 1-join, "number of books which have been read, commented and authored before" is a 2-join, and "number of books which have been read, commented, indicated as wish to read, and authored before" is a 3-join. For examples of a linear topology, "number of books which have been read by some users who have written some diary before" is a 1-join, "number of books which have been read by some users who have written some diaries about the same book" is a 2-join, and "number of books which have been read by some users who have written some diaries about the same book and also wrote some reviews" is a 3-join. Their corresponding SQL queries are presented in the full paper [31].

Since DSCALER replicates Referencing Vector Correlation in Step1d, we expect it to do well for star joins, as confirmed by Figure 27. For linear joins, however, Figure 27 indicates errors are big even for a 1-join, and they increase with n. This is also expected, since $CoDa_{\mathcal{D}}$ does not capture the chained correlation in a linear n-join.

(2) Social Networks

For `DoubanBook`, two users may form a social link via their posts. Consider a feature like `mutual-comment`, where user U_x makes x comments on posts written by user U_y, who makes y comments on posts written by U_x. Such correlation between U_x and U_y is not in CoDa. If z counts the number of such (x, y) pairs, the frequency distribution for z is plotted in Figure 28 for `diary` as posts, and in Figure 29 for `review`. They show the empirical distribution for D_2 is much more skewed than the distribution for $\widetilde{D_2}$ generated by DSCALER.

10. CONCLUSION

This paper introduces nuDSP, presents a solution DSCALER (with supporting Theorems 1–4) and uses 3 similarity measures to compare it to 2 algorithms for 3 datasets.

Recall from **I1** and **I7** (Section 1.2) that one could improve the similarity between $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ by adding more features into the 3-phase framework, but that would make the solution harder. Surely, it would not be *scalable* (pun intended) to design a new algorithm everytime a new feature is added to $\mathcal{F}$. Instead, our current work explores the possibility of developing a set of tools for tweaking $\widetilde{\mathcal{D}}$.

For example, to enforce similarity for linear joins in addition to CoDa, we want to generate $\widetilde{\mathcal{D}}$ by $\mathcal{T}_{\texttt{linear}}(\text{DSCALER}(\mathcal{D}, \mathbf{s}))$, for some tool $\mathcal{T}_{\texttt{linear}}$ that tweaks the DSCALER output. Similarly, to generate social links, we would run some tool $\mathcal{T}_{\texttt{social}}$ on the DSCALER output.

There are several challenges to this approach. For example, to get $\mathcal{T}_{\texttt{social}}(\mathcal{T}_{\texttt{linear}}(\text{DSCALER}(\mathcal{D}, \mathbf{s})))$, running $\mathcal{T}_{\texttt{social}}$ after $\mathcal{T}_{\texttt{linear}}$ may adversely perturb the correlation injected by $\mathcal{T}_{\texttt{linear}}$; moreover, $\mathcal{T}_{\texttt{social}}$ and $\mathcal{T}_{\texttt{linear}}$ may not commute, i.e. $\mathcal{T}_{\texttt{social}}(\mathcal{T}_{\texttt{linear}}(\text{DSCALER}(\mathcal{D}, \mathbf{s}))) \neq \mathcal{T}_{\texttt{linear}}(\mathcal{T}_{\texttt{social}}(\text{DSCALER}(\mathcal{D}, \mathbf{s})))$.

Our current work on developing a set of scaling tools is a step towards the vision for application-specific benchmark generation, as proposed in previous work [28].

11. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, pages 275–286, 1999.

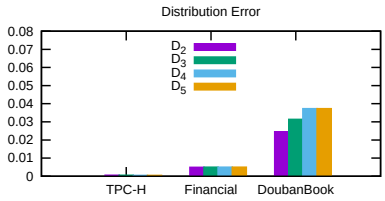

Figure 26: Distribution Error for Nonuniform Scaling

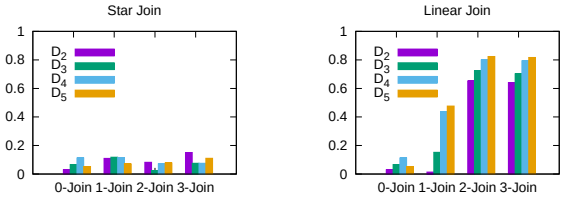

Figure 27: Query Error for Star Join and Linear Join

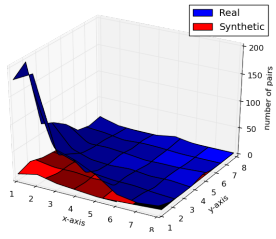

Figure 28: Mutual-comment for `diary`

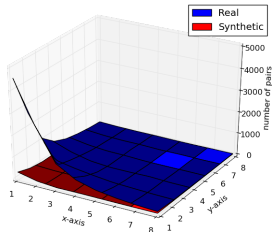

Figure 29: Mutual-comment for `review`

[2] S. Agarwal, A. P. Iyer, et al. Blink and it's done: interactive queries on very large data. *VLDB*, 5(12):1902–1905, 2012.

[3] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *SIGMOD*, pages 685–696, 2011.

[4] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In *ICDE*, pages 506–515. IEEE, 2007.

[5] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: generating query-aware test databases. In *SIGMOD*, pages 341–352, 2007.

[6] N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, pages 1097–1107, 2005.

[7] T. Buda, T. Cerqueus, et al. ReX: Extrapolating relational data in a representative way. In *Data Science*, LNCS 9147, pages 95–107. 2015.

[8] T. S. Buda, T. Cerqueus, et al. VFDS: An application to generate fast sample databases. In *CIKM*, pages 2048–2050, 2014.

[9] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD*, pages 287–298, 2004.

[10] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD*, pages 89–98. ACM, 2003.

[11] S. Duan, A. Kementsietsidis, et al. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *SIGMOD*, pages 145–156, 2011.

[12] H. Fu, A. Zhang, and X. Xie. Effective social graph deanonymization based on graph structure and descriptive information. *ACM Trans. Intell. Syst. Technol.*, 6(4):49:1–49:29, July 2015.

[13] R. Gemulla, P. Rösch, and W. Lehner. Linked Bernoulli synopses: Sampling along foreign keys. In *Scientific and Statistical Database Management*, pages 6–23, 2008.

[14] J. Gray, P. Sundaresan, et al. Quickly generating billion-record synthetic databases. In *SIGMOD*, pages 243–252, 1994.

[15] V. Gupta, G. Miklau, and N. Polyzotis. Private database synthesis for outsourced system evaluation. In *Proc. AMW*, May 2011.

[16] K. Houkjær, K. Torp, and R. Wind. Simple and realistic data generation. In *VLDB*, pages 1243–1246, 2006.

[17] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *KDD*, volume 96, pages 367–370, 1996.

[18] H. Köhler, X. Zhou, et al. Sampling dirty data for matching attributes. In *SIGMOD*, pages 63–74, 2010.

[19] E. Lo, N. Cheng, and W.-K. Hon. Generating databases for query workloads. *PVLDB*, 3(1-2):848–859, 2010.

[20] W. Lu, G. Miklau, and V. Gupta. Generating private synthetic databases for untrusted system evaluation. In *ICDE*, pages 652–663, Mar. 2014.

[21] F. McSherry and I. Mironov. Differentially private recommender systems. In *KDD*, pages 627–636, 2009.

[22] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symp. Security and Privacy*, pages 111–125, 2008.

[23] C. R. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. *SIGMOD Rec.*, 29(2):82–92, May 2000.

[24] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *SIGMOD*, pages 23–32, 1999.

[25] S. Qiao and Z. M. Özsoyoğlu. RBench: Application-specific RDF benchmarking. In *SIGMOD*, pages 1825–1838, 2015.

[26] T. Rabl, M. Danisch, et al. Just can't get enough: Synthesizing big data. In *SIGMOD*, pages 1457–1462, 2015.

[27] J. M. Stephens and M. Poess. MUDD: a multi-dimensional data generator. In *SIGSOFT Software Engineering Notes*, pages 104–109, 2004.

[28] Y. C. Tay. Data generation for application-specific benchmarking. *PVLDB*, 4(12):1470–1473, 2011.

[29] Y. C. Tay, B. T. Dai, et al. UpSizeR: Synthetically scaling an empirical relational database. *Inf. Syst.*, 38(8):1168–1183, 2013.

[30] X. Yin, J. Han, et al. Efficient classification across multiple database relations: A CrossMine approach. *IEEE TKDE*, 18(6):770–783, 2006.

[31] J. W. Zhang and Y. C. Tay. Dscaler: Synthetically scaling a given relational database. `http://www.comp.nus.edu.sg/~upsizer/, 2016.`