Stochastic Submodular Data Forgetting

Ramon Rico
Utrecht University
Utrecht, The Netherlands
r.ricocuevas@uu.nl

Arno Siebes
Utrecht University
Utrecht, The Netherlands
a.p.j.m.siebes@uu.nl

Yannis Velegrakis
U. of Trento and Utrecht University
Utrecht, The Netherlands
i.velegrakis@uu.nl

Abstract

Our ability to collect data is rapidly surpassing our ability to store it. As a result, organizations are faced with difficult decisions about which data to retain and which to dispose of. Data forgetting, frames this reduction task as a subset selection exercise. Given a relational dataset D, a query log Q, and a budget B, the goal is to find a subset $D^* \subseteq D$ with at most B tuples such that it is still possible to compute, based solely on D^*, approximate answers to the expected query workload. Existing data forgetting routines have substantial limitations. They either offer strong theoretical guarantees but lack scalability due to function evaluation (submodular-based), or achieve scalability by avoiding function evaluation but lack theoretical guarantees (amnesia-based). To bridge the gap between the limitations of submodular and amnesia based methods, we propose `IndepDF` and `DepDF`: two data forgetting routines that offer scalability by avoiding function evaluation while maintaining strong theoretical guarantees. Our extensive experimental evaluation on real and synthetic datasets demonstrates that our algorithms are capable of matching the performance of the state-of-the-art submodular-based routines while exhibiting a runtime comparable to that of amnesia-based algorithms. In essence, combining the best traits of both.

CCS Concepts

• **Theory of computation → Data structures and algorithms for data management**; • **Mathematics of computing → Continuous optimization**.

Keywords

Data forgetting, Submodular optimization, Data reduction, Data cleaning, Data quality, Data sustainability, Big data.

ACM Reference Format:
Ramon Rico, Arno Siebes, and Yannis Velegrakis. 2025. Stochastic Submodular Data Forgetting. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 365 (December 2025), 15 pages. https://doi.org/10.1145/3769830

1 Introduction

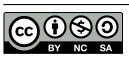
Over the past decade, we have witnessed an unprecedented data-driven revolution that has transformed nearly every aspect of our lives. Massive volumes of data are being collected, processed, integrated, and analyzed, driving crucial advancements in fields such as logistics, medicine, and science [8]. Nevertheless, this revolution encounters significant challenges. While advances in storage technology have traditionally allowed organizations to accumulate data with almost no restriction, the size of the global datasphere has recently outpaced storage production [35]. More alarming, the annual storage production is expected to fall below 1% of the data produced annually by the year 2030 [16]. That is, assuming that old data is not deleted, we will only be able to store less than 1% of all the new annually created data. Furthermore, uncontrolled data storage often leads to "dirty" IT environments where redundant or obsolete data is accumulated, compromising the integrity of data retrieval processes and knowledge discovery algorithms.

To effectively address these challenges, data-driven enterprises and research institutes are forced to undertake *data reduction*: retaining the information hidden in their data while respecting regulatory, storage, and processing constraints [28]. In other words, they must decide what data to keep and in what form in order to comply with legal data regulations and storage restrictions, while minimizing information loss. Furthermore, due to the exponential-growing nature of the global datasphere over the last decade[1] automating the reduction exercise becomes vital to avoid data flooding [21].

Existing data reduction techniques reduce large datasets by retaining their most relevant fragments based on a notion of data importance. One of the most popular approaches for dataset reduction is *data summarization* [2, 5–7, 25, 34, 37–39], where data importance is understood as data representativeness. In other words, the most important datapoints in a dataset are those that collectively best represent the *complete* dataset. *Data rotting* [19, 20] was first to challenge this notion of data importance by proposing that data "rots-away", losing value over time; and hence, data reduction techniques should rely on notions of data importance orthogonal to representativeness. As a result, a new generation of data reduction routines that understand data importance as *data-usage* emerged. Particularly, query based *amnesia* algorithms [21] and *submodular* based methods [9, 10, 13] deem the most important data points in a dataset to be those that collectively best represent the *queried / retrieved* portions of the database.

We identify three limitations in the existing literature. First, there is no general mathematical formulation for this new usage-driven data reduction paradigm. Second, submodular based routines [9, 10, 13] enable dataset reduction by producing solutions with strong theoretical quality guarantees. However, they often become infeasible in data-intensive scenarios due to the need for iteratively evaluating expensive optimization objectives (referred to as function evaluation). Third, amnesia algorithms [21] do not face these scalability issues

Authors' Contact Information: Ramon Rico, Utrecht University, Utrecht, The Netherlands, r.ricocuevas@uu.nl; Arno Siebes, Utrecht University, Utrecht, The Netherlands, a.p.j.m.siebes@uu.nl; Yannis Velegrakis, U. of Trento and Utrecht University, Utrecht, The Netherlands, i.velegrakis@uu.nl.



ACM 2836-6573/2025/12-ART365
https://doi.org/10.1145/3769830

[1] ref. IDC White Paper (Seagate)), https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf

since they do not rely on function evaluation. However, they produce solutions that lack theoretical quality guarantees. To address these limitations, we offer two core contributions:

First, we propose what is, to the best of our knowledge, the first stochastic and topology-aware formulation for this new usage-driven data reduction paradigm, unifying the ideas presented in [8–10, 13, 21] under the same formal framework. We suggest naming this emerging data reduction paradigm *data forgetting*. The inspiration behind this name choice stems from the vision paper [21], which proposes that data management systems should not store records indefinitely but rather have the capability to selectively "forget" entries that have become valueless. We formalize data forgetting as an optimization problem (Problem 1): Given a relational database D and a set of monotone queries Q distributed according to a probability distribution $\mathcal{Q}$, find a succinct subset of tuples $D^* \subseteq D$ that allows to faithfully answer queries sampled from distribution $\mathcal{Q}$.

Second, we propose `DepDF` and `IndepDF`: two data forgetting algorithms that avoid function evaluation while providing strong theoretical quality guarantees. This approach effectively bridges the gap between the limitations of submodular-based routines and amnesia algorithms. In this sense, our routines combine the best of both worlds: scalability from avoiding function evaluation and strong theoretical quality guarantees.

`DepDF` extends the stochastic submodular maximization framework [14, 17, 31] with a novel double-stochastic gradient estimation. More concretely, `DepDF` lifts the data forgetting problem into the continuous domain with the multilinear extension [41]. Then, it finds a continuous solution iteratively via SCG^2, a novel efficiency-improved version of SCG [31], which we attain by leveraging our proposed novel double-stochastic gradient estimation. Last, it brings the solution back into the discrete domain via pipage rounding [3], attaining a solution with an approximation factor of $(1 - \frac{1}{e} - \epsilon)$, in expectation. Our proposed doubly-stochastic gradient estimation, which we theoretically justify in Theorem 5.2, allows to replace the costly function evaluation by a fast sampling step that only requires accessing one query $q \in Q$ and one datapoint $d \in q(D)$ per iteration, followed by the evaluation of a simple proxy objective, described in Equation 6. In practice, our proposed double-stochastic gradient estimation allows `DepDF` to produce solutions of equivalent quality to those generated by the state-of-the-art submodular maximization algorithms in less than 1% of the time. Using continuous methods to solve discrete problems at scale is a well-founded approach. It has enabled designing scalable solutions for other discrete problems such as pattern set mining [11].

`IndepDF` is a deterministic routine that assigns a value to each datapoint in the database and simply retains the top-scoring points. Despite its simplicity, it can provide an exact solution in polynomial time for the data forgetting problem under the setup described in the original vision paper [21]. We formally prove this theoretical finding in Theorem 5.6. Despite its limited theoretical guarantees for the general data forgetting problem, `IndepDF` enjoys strong data-dependent guarantees. We show experimentally that it provides high quality solutions when queries retrieve homogeneous views of the database. Specifically, in this seeting, it is capable of producing solutions that match those generated by the state-of-the-art submodular maximization algorithms in less than 0.01% of the time.

Summarizing, our contributions are:

- A unifying stochastic and topology-aware formulation for the new data reduction paradigm, where data importance is understood as data usage, which we propose to name *data forgetting*.
- `DepDF`: An approximation algorithm to solve data forgetting by extending the stochastic submodular maximization framework with a novel double-stochastic gradient estimation. We theoretically justify this choice in Theorem 5.2. `DepDF` provides strong theoretical guarantees while maintaining scalability due to function evaluation avoidance.
- `IndepDF`: An exact algorithm to solve data forgetting in a simplified setting using a straightforward approach. We formally prove the algorithm's exactness in Theorem 5.6. Despite its limited theoretical guarantees for the general data forgetting problem, we show experimentally that it provides high quality solutions when queries retrieve homogeneous views of the database. Similarly to `DepDF`, `IndepDF` also enjoys scalability due to function evaluation avoidance.
- An extensive comparative evaluation of the two algorithms against the state-of-the-art along with a scalability study, offering valuable insights.

The remainder of the paper is structured as follows: Section 2 presents the related work. Section 3 introduces the necessary notation and Section 4 formulates the data forgetting problem. Section 5 introduces formally `DepDF` and `IndepDF` alongside a comparative complexity analysis against the state-of-the-art, while Section 6 provides an extensive experimental evaluation.

2 Related Work

2.1 Data Summarization

Of the most popular approaches for dataset reduction is *data summarization* [2, 5–7, 25, 34, 37–39]. A *summary* of a dataset D is a brief synopsis that has a modest size compared to that of the latter. Summaries come in one of two flavours: as subsets of D, or as representative values that replace the original data records. Further, summarization algorithms fall into one of three categories: statistical [2, 5, 39], submodular [25, 37, 38], or geometric [6, 7, 34].

Despite being of different nature, all summarization techniques share a similar philosophy when addressing data reduction: every datapoint in the input dataset D is assumed to convey a certain degree of valuable information. This is simply because summarization routines conceive data importance as data representativeness. In other words, they deem the most important datapoints in a dataset those that collectively best represent the *complete* dataset. *Data rotting* [19, 20] was the first to challenge this premise by proposing that data, like everything else in nature, "rots away" losing its value over time. Hence, data reduction techniques should rely on notions of data-importance orthogonal data representativeness.

2.2 A New Generation of Data Reduction Techniques

As a response to the challenges raised by the data rotting paradigm, a new generation of data reduction routines that understand data importance as *data-usage* emerged. We name this emerging data reduction paradigm as *data forgetting*. The inspiration behind our

name choice stems from a recent vision paper [21], which proposes that data management systems should not store records indefinitely but rather have the capability to selectively "forget" entries that have become valueless. Data forgetting routines aim to identify the most queried / retrieved regions within a dataset and "forget" (i.e., delete) everything that lies outside of them. Effectively, re-defining the notion of data-value: the most important data points in a database are no longer those that collectively best represent the complete database, but rather those that collectively best represent the used/accessed fractions of the database. To date, only two existing techniques fall under this paradigm: amnesia algorithms [21] and submodular-based routines [9, 10, 13].

Amnesia algorithms [21] are probabilistic routines that provide databases the power to controllably forget data entries over time. Their output's quality is evaluated on the extent to which expected query precision is maximized. Nevertheless, a significant limitation of these routines lies in the absence of theoretical guarantees. For the contrary, our most light-weight algorithm, `IndepDF`, is capable of yielding an *exact* solution in polynomial time in this setting. We formally prove the algorithm's exactness in Theorem 5.6.

Submodular-based routines exploit modifications of the classic `GREEDY` algorithm [32] to sieve subsets that maximize topology-aware versions of query satisfiability. For instance, Gershtein et al. [13] propose using the `LAZY GREEDY` algorithm for forgetting e-commerce data. The `LAZY GREEDY` algorithm, originally introduced by Minoux [29], exploits the submodularity of the optimization objective to speed up `GREEDY` via lazy evaluation. Recently, Davidson et al. [9, 10] proposed using fast approximate nearest-neighbor computations to speed up function evaluation in `LAZY GREEDY` when forgetting image data. More precisely, they resort on LSH to only consider similarities between image pairs corresponding to hash collisions.

In spite of the ability of submodular algorithms to achieve high quality solutions when forgetting data, a noteworthy limitation of these routines resides in the necessity for function evaluation. Submodular algorithms like `GREEDY` or `LAZY GREEDY`, require $O(n \cdot B)$ evaluations of the optimization objective to extract a solution with B elements out of a dataset with n records. Consequently, in settings like data forgetting, where evaluating the objective is costly, these algorithms become infeasible. Refer to Section 5.2 for more details about the complexity analysis of submodular algorithms on the data forgetting problem. To overcome this limitation, our proposed algorithm, `DepDF`, avoids function evaluation by extending the stochastic submodular maximization framework with a novel double-stochastic gradient estimation.

2.3 Materialized View Selection

At a high level, data forgetting bears similarities with *materialized view selection* (MVS) [1, 27, 30, 36, 40]. The goal of MVS is to select a set of views to materialize, striking the best balance between query processing cost, view maintenance cost and storage space [27]. Just like in data forgetting, submodular-based techniques have been proposed to address materialized view selection [18]. However, there exist notable differences between MVS and data forgetting. While data forgetting allows partially incomplete query answer sets by relying on a notion of pair-wise tuple similarity, a materialized view always contains the complete answer set of a given query. Moreover, MVS concerns pre-computing and caching query results, while data forgetting involves removing some of the data entries over which the query results are computed.

2.4 Approximate Query Processing

At a high level, data forgetting also shares some traits with *approximate query processing* (AQP) [4, 12, 24, 26]. Specifically, both involve providing approximate answers to a workload of queries. However, there exist notable differences between AQP and data forgetting. The goal of AQP is to provide approximate answers with acceptable accuracy in orders of magnitude less query response time than that for the exact query processing [26]. AQP is typically attained by building and dynamically maintaining appropriate data structures that allow retrieving approximate answer sets in a timely manner. For the contrary, the goal of data forgetting is to select which portions of the database to delete such that exact query processing is minimally compromised.

3 Preliminaries

Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a collection of attributes, each associated with a finite domain Dom_{A_i}. A **tuple** is an $m-$dimensional vector $d = (v_1, \ldots, v_m)$ where $v_i \in Dom_{A_i}$. The set $\mathfrak{Constrs}(d) = \{A_1 = v_1, \ldots, A_m = v_m\}$ is the set of **conditions** of d. The set of all possible tuples $\mathcal{U} = Dom_{A_1} \times \cdots \times Dom_{A_m}$ constitutes the **universe**. A **dataset** is a set $D \subseteq \mathcal{U}$ (i.e., a finite collection of tuples). In other words, a dataset D is a single table with a finite number of rows and a finite number of columns. Namely, $|D| = n \in \mathbb{N}$ rows and $|\mathcal{A}| = m \in \mathbb{N}$ columns. To ensure that a dataset is truly a set, we assume the existence of unique tuple identifiers.

Let q denote a **query**. The **answer set** of a query q on a dataset D, denoted as $q(D)$, is the set of tuples in D that satisfy q. Given a query q and datasets $D' \subseteq D \subseteq \mathcal{U}$, the **precision** of q in D' with respect to D, is the fraction of tuples shared by answer sets $q(D')$ and $q(D)$. Formally,

$$Precision(q, D', D) = \frac{|q(D')|}{|q(D)|} \in [0, 1].$$

A query q is **monotone** if for all datasets $D' \subseteq D \subseteq \mathcal{U} \implies q(D') \subseteq q(D)$. Henceforth, the term **query** shall denote monotone queries exclusively.

Given a dataset $D \subseteq \mathcal{U}$, a **set function** $f : \mathcal{P}(D) \mapsto \mathbb{R}$ is:

- **non-negative** if $\forall\, A \subseteq D$, $f(A) \geq 0$.
- **monotone** if $\forall\, A \subseteq B \subseteq D$, $f(A) \leq f(B)$.
- **submodular** if $\forall\, A \subseteq B \subseteq D$, and $d \in D \setminus B$,

$$\Delta_f(d|B) \leq \Delta_f(d|A),$$

where $\Delta_f(d|D') := f(D' \cup \{d\}) - f(D')$ is the discrete derivative of f at $D' \subseteq D$ with respect to d.

Submodularity is a diminishing returns property that resembles a discrete version of concavity. A function f is submodular if given nested subsets $A \subseteq B \subseteq D$ and a datapoint $d \in D \setminus B$, the marginal gain of including d in the bigger one (i.e., in B) is not greater than the marginal gain of including it in the smaller one (i.e., in A) [22].

4 Data forgetting

Data forgetting is the novel approach for big data reduction that defines data importance based on data usage. Given a dataset D alongside structured information about its past usage, data forgetting involves reducing D by identifying the regions of the dataset that are expected to be used in the future, and deleting everything that lies outside of them. Typically, information about past data usage comes in the form of a log of queries Q that retrieve different portions of D.

Data forgetting is dynamic by nature. Tuples and queries arrive in a streaming fashion. Newly arrived tuples get stored in the database D, while newly received queries get logged in the query-log Q. When the storage constraint is reached, space must be freed to accommodate newly arriving tuples. The *forgetting kernel* takes care of forgetting (i.e., deleting) the unimportant regions of D. More precisely, fixed a budget B, the forgetting kernel runs a data forgetting algorithm that selects a subset $D^* \subseteq D$ with at most B tuples. Subsequently, tuple set D^* is retained and tuple set $D \setminus D^*$ is forgotten. We refer to this subset selection exercise as a *data forgetting round*. Figure 1 depicts the pipeline of a data forgetting round. A DBMS can execute a data forgetting round at any point in time. Generally, it is sensible to do so when the storage constraint is met or close to being met.

Data forgetting could have more dimensions to remove unnecessary information (e.g., removing columns instead of rows). Notably, dimensionality reduction techniques are orthogonal to row-level data forgetting. Techniques like Principal Component Analysis (PCA) [33] that offer an alternative (dimensionality-reduced) representation of D can be applied before a row-level data forgetting routine to further minimize storage needs.

As aforementioned, data forgetting defines data importance based on data usage. By default, the latter is highly dependent on the specific queries being run. Hence, there is always a risk that data entries that were removed in past data forgetting rounds will become important for future queries. However, while future queries may introduce new patterns, the up-to-date query-log Q is the only reliable source of information that we have about the actual usage of the database. Therefore, arguably, any decision about data removal must be grounded in this log. Without concrete evidence of future usage patterns, it is impractical to anticipate the needs of unknown queries. We can only make decisions based on observed queries, and thus, the up-to-date query-log Q is the most reasonable tool to guide data reduction exercise. There are many types of queries in a DBMS. Following related works [8–10, 13, 21], we restrict ourselves to monotone queries.

Consequently, we formalize a data forgetting round as the following optimization problem: Given a dataset D and a set of monotone queries Q distributed according to a probability distribution $\mathcal{Q}$, the goal is finding a subset of D with significantly fewer elements that still allows coping with the expected query workload (based on distribution $\mathcal{Q}$). Stated differently, finding a subset $D^* \subseteq D$ composed of at most $B < |D|$ tuples for which the most likely queries $q \in Q$ according to distribution $\mathcal{Q}$, have reduced answer sets $q(D^*)$ of maximal resemblance to their complete counterparts $q(D)$. More precisely,

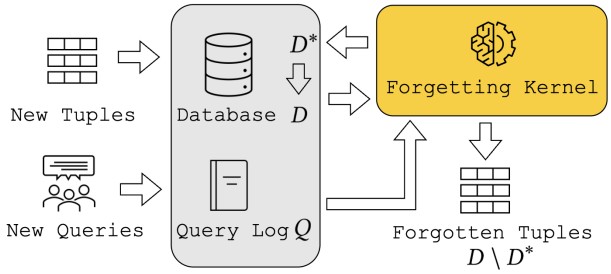


Figure 1: Data forgetting round.

PROBLEM 1 (**DATA FORGETTING** (ROUND)). *Given:*

- *A dataset $D = \{d_1, \ldots, d_n\}$;*
- *A set of monotone queries Q, along with a probability distribution $\mathcal{Q}$ defined on Q;*
- *A set similarity function $S : \mathcal{P}(D) \times \mathcal{P}(D) \mapsto \mathbb{R}_+$;*
- *A budget $B \in \mathbb{N}$;*

find a subset $D^* \subseteq D$ such that

$$D^* = \underset{D' \subseteq D,\ |D'| \leq B}{\arg\max} \quad f(D')$$

where

$$f(D') = \mathbb{E}_{q \sim \mathcal{Q}}[S(q(D), q(D'))]. \tag{1}$$

Our formalization of the data forgetting problem (Problem 1) is novel. Unlike the formulations presented in related works [8–10, 13, 21], ours is both stochastic and topology-aware. That is, the optimization objective f in eq. (1) involves both a probability distribution $\mathcal{Q}$ over the set of queries Q and a topology-aware set-similarity function $S(\cdot, \cdot)$ to account for the topology of the data. The data forgetting optimization objective f captures the expected similarity between the complete and reduced answer sets ($q(D)$ and $q(D')$, respectively) for a query q drawn from distribution $\mathcal{Q}$. The distribution $\mathcal{Q}$ reflects the relative frequency of usage of each query $q \in Q$, and consequently, usually corresponds to the empirical distribution built from a query-log with records in Q. Alternatively, Q can also be a set of parameterizable queries or query templates, and $\mathcal{Q}$ represent the probability distribution over the parameter space. Further, set similarity function S allows quantifying the degree of discrepancy between the query results achieved on the complete dataset D and those obtained on the reduced dataset D' (i.e., between $q(D)$ and $q(D')$). Since the considered queries are monotone, $q(D') \subseteq q(D)$. That is, the reduced answer sets will *always* be subsets of their complete counterparts. Subset-set similarity is typically defined in terms of overall diversity and coverage [25, 37, 38]. Stated differently, $q(D')$ and $q(D)$ are deemed similar if the former provides maximal diverse coverage over the later.

There exists a plethora of subset-set similarity mappings that mathematically crystallize the notions of overall diversity and coverage. These functions have been used as objectives for related data reduction tasks, such as scene, document, and image collection summarization. Among these, the *facility location* function stands out as the most well-motivated and widely used optimization objective [25, 37, 38]. Formally, $\forall\ D' \subseteq D$, $f_{\text{facility location}}(D') := \sum_{d \in D} \max_{d' \in D'} \text{sim}(d, d')$. Further, the "query-aware" version of

this function has been used to measure the similarity between reduced and complete answer sets in the context of select queries in image data [9, 10]. Hence, we propose using a normalized "query-aware" version of the facility location function to measure the similarity between reduced and complete answer sets for monotone queries. Formally, given $D' \subseteq D$ and $q \in Q$, we define

$$S(q(D), q(D')) := \frac{1}{|q(D)|} \cdot \sum_{d \in q(D)} \max_{d' \in q(D')} \text{sim}(d, d'), \quad (2)$$

where $\text{sim}(\cdot, \cdot)$ indicates the (point-wise) similarity between any given pair of data points. If D is categorical, it is natural to consider

$$\text{sim}(d, d') = Jaccard(d, d') := \frac{|\mathfrak{Constrs}(d) \cap \mathfrak{Constrs}(d')|}{|\mathfrak{Constrs}(d) \cup \mathfrak{Constrs}(d')|}, \quad (3)$$

where $\mathfrak{Constrs}(\cdot)$ is defined as in Section 3. On the contrary, if D is numerical, it is reasonable choosing

$$\text{sim}(d, d') = \cos(d, d') := \frac{d \cdot d'}{||d|| \cdot ||d'||}. \quad (4)$$

In the case that the dataset D contains both categorical and numerical values, two effective approaches are either to discretize the numerical values and use the Jaccard similarity (eq. 3), or to embed the categorical values and use the Cosine similarity (eq. 4). Further, if a query q includes a projection π, the point-wise similarity between datapoints d and d' (as defined in Equations 3 and 4) is measured between their projected counterparts $\pi(d)$ and $\pi(d')$. Nonetheless, the formulas are presented in the no-projection case for notational simplicity. Further, Equation 4 is typically normalized when d and d' have negative entries.

According to the normalized query-aware facility location mapping (equation 2), the set similarity between answer sets $q(D)$ and $q(D')$ is the sum of the point-wise similarities between each element in $q(D)$ and its closest neighbor inside $q(D')$. Note that the nearest neighbor of any point $d \in q(D)$ inside of $q(D')$ will always be an element of $q(D)$ itself. This is because, since the queries are assumed monotone, $q(D') \subseteq q(D)$. Hence, function S as in equation 2 is maximal when $q(D')$ is composed by those elements in $q(D)$ that most effectively maintain the cluster structure of $q(D)$. The data forgetting optimization objective f, simply weights the individual answer set similarity contributions computed according to normalized query-aware facility location mapping S via the probability distribution $\mathcal{Q}$.

EXAMPLE 4.1. *Consider the dataset D in Table 1 and assume that only two tuples can be retained (i.e., the budget B = 2). Additionally, suppose the only information available about the usage of D is the set of queries*

$$\begin{aligned} Q = \{\ & q_1 : \texttt{SELECT} * \texttt{FROM}\ D\ \texttt{WHERE}\ Age \geq 25 \\ & \quad \texttt{AND}\ Children < 3, \\ & q_2 : \texttt{SELECT} * \texttt{FROM}\ D\ \texttt{WHERE}\ City = Amsterdam \\ & \quad \texttt{AND}\ Job = Teacher, \\ & q_3 : \texttt{SELECT} * \texttt{FROM}\ D\ \texttt{WHERE}\ Name = Olivia\ \}, \end{aligned}$$

where $\mathbb{P}_{\mathcal{Q}}(q_1) = \mathbb{P}_{\mathcal{Q}}(q_2) = \mathbb{P}_{\mathcal{Q}}(q_3) = 1/3$. That is, three monotone queries occurring with uniform probability, each retrieving a different portion of D. More precisely, the answer sets of the queries q_1, q_2 and q_3 are $q_1(D) = \{d_1, d_2, d_3\}, q_2(D) = \{d_1, d_3\}$ and $q_3(D) = \{d_3\}$, respectively.

In light of these constraints, "forgetting" D involves retaining the tuple or tuple pair $D^* \subseteq D$ that best allows computing approximate answers to the expected query workload according to query distribution $\mathcal{Q}$. That is, keeping a subset $D^* \subseteq D$ with $|D^*| \leq 2$ such that for any query q drawn from distribution $\mathcal{Q} = unif\{q_1, q_2, q_3\}$, the reduced answer sets $q(D^*)$ offer a faithful representation of their complete versions $q(D)$.

Due to the budget constraint $B = 2$, one tuple must be forgotten. There exist three possible choices: Forgetting d_1, d_2, or d_3. Each choice corresponds, respectively, to retaining subset $D^* = \{d_2, d_3\}$, $D^* = \{d_1, d_3\}$, or $D^* = \{d_1, d_2\}$.

Forgetting d_1 would cause two queries to be incorrect since $q_1(D^*) = \{d_2, d_3\} \neq D = q_1(D)$, and $q_2(D^*) = \{d_3\} \neq \{d_1, d_3\} = q_2(D)$. Further, forgetting d_2 would cause one one query to be incorrect as $q_1(D^*) = \{d_1, d_3\} \neq D = q_1(D)$. Last, forgetting d_3 would cause three queries to be incorrect since $q_1(D^*) = \{d_1, d_2\} \neq D = q_1(D)$, $q_2(D^*) = \{d_1\} \neq \{d_1, d_3\} = q_2(D)$, and $q_3(D^*) = \emptyset \neq \{d_3\} = q_3(D)$.

A natural solution is forgetting d_2, the tuple that causes less queries to be incorrect. However, by making this choice we ignore the topology of the data D when observed under the lens of query-log Q. That is, we overlook the similarities between the tuples that are collectively retrieved by the queries in the query-log Q. Based on the given query workload we can make two key observations: First, d_1 is always returned alongside d_3. That is, the two answer sets $q_1(D) = D$ and $q_2(D) = \{d_1, d_3\}$ that contain d_1 also contain d_3. The converse is not true as $q_3(D) = \{d_3\}$. Second, every query is of equal importance since distribution $\mathcal{Q}$ is uniform.

Moreover, the pair-wise Jaccard similarity (eq. 3) between the tuples in D is $sim(d_1, d_3) = 6/8 = 0.75$, $sim(d_1, d_2) = sim(d_3, d_2) = 1/13 = 0.08$. Consequently, if both d_1 and d_3 are returned by a query it is sufficient to just keep one of them as they are highly similar. Further, if d_1 and d_2 are returned together or d_3 and d_2 are returned together, in order to preserve as much information as possible about the complete answer set, it is necessary to keep both as they are highly dissimilar.

Consequently, if we forget d_1, $q_1(D^*)$ contains $(0.75+1+1)/3 \approx 92\%$ of the information of $q_1(D)$, $q_2(D^*)$ contains $(0.75 + 1)/2 \approx 88\%$ of the information in $q_2(D)$, and $q_3(D^*)$ contains $1/1 = 100\%$ of the information in $q_3(D)$. Since every query is of equal importance due to the uniform distribution, forgetting d_1 yields reduced answer sets that contain on expectation $(0.92 + 0.88 + 1)/3 \approx 93\%$ of the information contained in their complete counterparts. A similar analysis reveals that forgetting d_2, yields reduced answer sets that contain on expectation $((1 + 0.08 + 1)/3 + (1 + 1)/2 + 1/1)/3 \approx 89\%$ of the information contained in their complete counterparts. Last, forgetting d_3, yields reduced answer sets that contain on expectation $((1+1+0.75)/3+(1+0.75)/2+0.75/1)/3 \approx 85\%$ of the information contained in their complete counterparts. In other words, forgetting d_1, d_2, d_3 result, respectively, in the preservation of $93\%, 89\%, 85\%$ of the information in D that is expected to be retrieved in base of

the usage-information provided by query set Q and distribution $\mathcal{Q}$. Therefore, forgetting d_1 is the best choice.

Table 1: Example dataset $D = \{d_1, d_2, d_3\}$.

id	*Name*	*Last name*	*Age*	*City*	*Job*	*Married*	*Children*
d_1	*James*	*Smith*	25	*Amsterdam*	*Teacher*	*Yes*	2
d_2	*Wade*	*Brown*	25	*Paris*	*Firefighter*	*No*	0
d_3	*Olivia*	*Smith*	25	*Amsterdam*	*Teacher*	*Yes*	2

5 Stochastic Submodular Maximization Framework

The Data Forgetting challenge, as it was defined earlier (ref. Problem 1), involves maximizing the set function $f(D')$, defined in Equation 1, under a cardinality constraint, namely, $|D'| \leq B$. Generally, maximizing an arbitrary set function under a cardinality constraint is intractable [22]. However, the data forgetting optimization objective has three properties that make the maximization exercise tractable.

THEOREM 5.1. *The set function $f(D')$ defined in Equation 1 is non-negative, monotone, and submodular.*

A consequence of the above theorem is that the Problem 1 admits a solution with $(1 - 1/e)$ approximation guarantee to the optimal one in polynomial time. Details of the proof can be found in the extended version of this paper. Such a solution can be obtained by exploiting any state-of-the-art submodular maximization routine, such as `GREEDY` [32] or `LAZY GREEDY` [29]. Nevertheless, a significant limitation of submodular methods lies in the need for objective evaluation. Submodular algorithms like `GREEDY` or `LAZY GREEDY`, require $O(n \cdot B)$ evaluations of the optimization objective f to extract a solution D^* of size B out of a dataset D of size n. Consequently, in settings like data forgetting, where evaluating the objective f is costly, these algorithms become infeasible.

To overcome this limitation, we propose maximizing f by extending the stochastic submodular maximization framework with a *double-stochastic* gradient estimation of F. F denotes the multilinear extension of f. In Theorem 5.2, we theoretically justify this choice and prove its efficiency over that of the default *single-stochastic* gradient estimation of F.

The vanilla stochastic submodular maximization framework consists of three stages. First, the problem is lifted to the continuous domain via the *multilinear extension* [41]. Next, a continuous solution is fetched by optimizing the multilinear extension [14, 17, 31]. Finally, the continuous solution is mapped back to the discrete domain using a value preserving rounding technique, typically *pipage rounding* or *randomized pipage rounding* [3]. This three-step framework allows maximizing objectives that are non-negative, monotone, submodular, and stochastic, without evaluating them directly. According to Theorem 5.1, the data forgetting objective is non-negative, monotone, submodular. Further, the stochasticity of the objective stems from the presence of the expected value over the probability distribution $\mathcal{Q}$ (i.e., operator $\mathbb{E}_{q\sim\mathcal{Q}}[\cdot]$ in Equation 1). In general, any objective function satisfying these four properties is amenable to be maximized via the stochastic submodular maximization framework.

When addressing data forgetting, the stochastic submodular maximization framework allows replacing the evaluation of the objective f with a sampling stage $q \sim \mathcal{Q}$ followed by the estimation of $\nabla F_q(\mathbf{x})$ (i.e., a *single-stochastic* estimate of $\nabla F(\mathbf{x})$). Here, $F_q(\mathbf{x})$ corresponds to the multilinear extension of $f_q(\cdot) = S(q(D), q(\cdot))$. Informally, the framework permits the maximization of f by "jumping over" the query set's expectation (i.e., "jumping over" $\mathbb{E}_{q\sim\mathcal{Q}}$). This is natural due to the linearity of the multilinear extension F and the linearity of the ∇ operator [41]: $\nabla F(\mathbf{x}) = \nabla\mathbb{E}_{q\sim\mathcal{Q}} F_q(\mathbf{x}) = \mathbb{E}_{q\sim\mathcal{Q}}\nabla F_q(\mathbf{x})$. Summing up, the vanilla stochastic submodular framework allows maximizing f by estimating $\nabla F_q(\mathbf{x})$: a *single-stochastic* estimate of $\nabla F(\mathbf{x})$. However, we theoretically prove that this process can be further optimized. Specifically:

THEOREM 5.2. *Let f denote the data forgetting optimization objective as defined in Equation 1, and F its multilinear extension. Then, the following properties hold:*

1. *Function f is doubly-stochastic. That is, there exists a probability distribution $\mathcal{D}$ and set functions $f_{q,d}(D')$ such that $f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}} f_{q,d}(D')$. Specifically, $\mathcal{D}$ is the probability distribution with probability mass function*

$$\mathbb{P}_{\mathcal{D}}(d, q) = \frac{|q(\{d\})|}{|q(D)|} \cdot \mathbb{P}_{\mathcal{Q}}(q),$$

and

$$f_{q,d}(D') = \max_{d'\in q(D')} sim(d, d').$$

2. *The gradient of the multilinear extension, $\nabla F(\mathbf{x})$, admits a double-stochastic estimation. Namely, $\nabla F_{q,d}(\mathbf{x})$ where $F_{q,d}(\mathbf{x})$ is the multilinear extension of $f_{q,d}(D')$.*
3. *Estimating $\nabla F_q(\mathbf{x})$ (the single-stochastic estimate of the gradient $\nabla F(\mathbf{x})$) requires estimating at least as many terms as estimating $\nabla F_{q,d}(\mathbf{x})$ (the double-stochastic estimate of the gradient $\nabla F(\mathbf{x})$).*

Details of the proof can be found in the extended version of this paper. Informally, Theorem 5.2 "unlocks" the maximization of the data forgetting objective f by not only "jumping over" the query set's expectation (as in the vanilla framework) but also "jumping over" the (hidden) answer set's expectation (i.e., "jumping over" $\mathbb{E}_{(d,q)\sim\mathcal{D}}$). We refer to the extension of the stochastic submodular maximization framework via the double-stochastic estimation of $\nabla F(\mathbf{x})$ as the `Dep`endent `D`ata `F`orgetting (`DepDF`) algorithm (Algorithm 1). Following, each step of the routine will be detailed.

Algorithm 1: `DepDF`

Data: Dataset D, queryset Q with distribution $\mathcal{Q}$, and budget $B \in \mathbb{N}$. **Hyperparameter**: Number of gradient ascent rounds $T \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ with $|D^*| \leq B$ optimizing $f =$ eq. (1) for $S =$ eq. (2) within $(1 - 1/e - O(T^{-1/3}))$ approximation factor.

$F \leftarrow$ `Multilinear_Extension`(f);
$\mathbf{x}^* \leftarrow$ `SCG`$^2(F, T)$ (Algorithm 2);
$\mathbf{e}_{D^*} \leftarrow$ `Randomized_Pipage_Rounding`$(\mathbf{x}^*)$;
return D^*

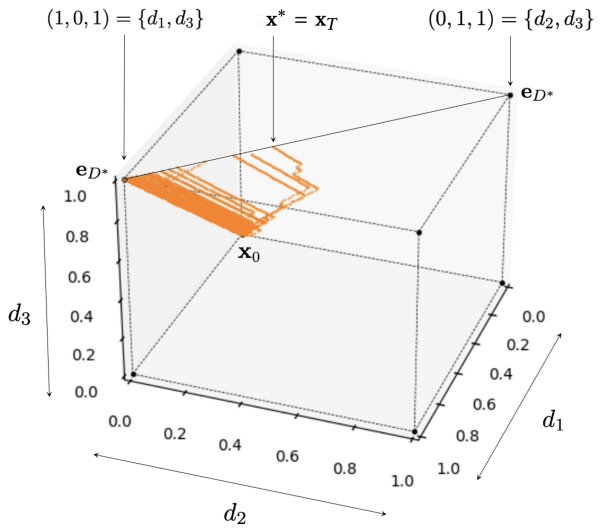


Figure 2: Visual representation of the solutions generated by 100 executions of `DepDF`. The orange trajectories starting from the origin depict the partial solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{100} = \mathbf{x}_T = \mathbf{x}^*$ generated the `SCG`^{2} calls (one call per execution of `DepDF`). Vector $\mathbf{e}_{D^*}$ denotes the outcome of the Pipage rounding scheme.

Multilinear extension. According to Theorem 5.1, the data forgetting objective f is non-negative, monotone, submodular. Hence, it is amenable to a multilinear extension [41]. The power set of D is identified with the corners of the unit hyper-cube $[0,1]^{|D|}$. That is, each subset of tuples $D' \subseteq D$ gets identified with a binary $|D|$-dimensional vector in $\mathbf{x} \in \mathbb{R}^{|D|}$ where $x_i = 1$ if $d_i \in D'$ and $x_i = 0$ if $d_i \notin D'$. The multilinear extension F takes the same values as f in the corners of the unit cube and extends it continuously through the cube's filling. Function F is linear, non-negative, monotone and DR-submodular [2]. The general mathematical definition of the multilinear extension can be found in [41]. In our case, exploiting the linearity of the multilinear extension, and property 1 in Theorem 5.2, it takes the form

$$F(\mathbf{x}) = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, F_{q,d}(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^{|D|}. \tag{5}$$

Example 5.3. *Revisiting Example 4.1, the power set of $D = \{d_1, d_2, d_3\}$ is identified with the corners of the unit cube in $\mathbb{R}^3$ as follows: $\emptyset = (0,0,0)$, $\{d_1\}$=$(1,0,0)$, $\{d_2\}$=$(0,1,0)$, $\{d_3\}$=$(0,0,1)$, $\{d_1, d_2\}$=$(1,1,0)$, $\{d_2, d_3\}$=$(0,1,1)$, $\{d_1, d_3\}$=$(1,0,1)$, and D=$(1,1,1)$. The multilinear extension F takes the same value as the discrete objective f on said corners and extends it to the remaining of the cube as indicated by Equation 5. For instance, $F(1,0,1) = f(\{d_1, d_3\}) = 0.8974$ and $F(0,1,1) = f(\{d_2, d_3\}) = 0.9306$. Further, F takes intermediate values that grow in a diminishing returns fashion from $(1,0,1)$ to $(0,1,1)$ along the diagonal line depicted in Figure 2.*

[2]A function F is DR-submodular if $\forall\ \mathbf{x}, \mathbf{y} \in [0,1]^n$, $\mathbf{x} \leq \mathbf{y} \implies \nabla F(\mathbf{y}) \leq \nabla F(\mathbf{x})$. Note that the definition of DR-submodularity is the continuous counterpart of the definition of submodularity. The discrete derivative is simply replaced with the gradient.

Finding a continuous solution. In order to fetch a continuous solution, we must optimize $F(\mathbf{x})$ over the convex set $C = \{\mathbf{x} \in [0,1]^{|D|} : \sum_{i=1}^{|D|} x_i \leq B\}$, where C is the continuous counterpart of the discrete storage constraint $D' \subseteq D$, $|D'| \leq B$. For this matter, we propose extending the *Stochastic Continuous Greedy* algorithm (SCG) [31] via our proposed double-stochastic estimation of ∇F, described in property 2 of Theorem 5.2. We denote this extended version `SCG` as `SCG`^{2} (Algorithm 2). The superscript 2 stands for the *double*-stochastic estimation of ∇F.

As opposed to `SCG`, which requires access to unbiased estimates of $\nabla F_q(\mathbf{x})$ for $q \sim Q$ (i.e., a single-stochastic estimate of $\nabla F(\mathbf{x})$), `SCG`^{2} only requires access to unbiased estimates of $\nabla F_{q,d}(\mathbf{x})$ for $(d,q) \sim \mathcal{D}$ (i.e., a doubly-stochastic estimate of $\nabla F(\mathbf{x})$). As a result of property 3 in Theorem 5.2, `SCG`^{2} is more efficient than `SCG`.

Using the product rule to sample $(d,q) \sim \mathcal{D}$, we derive an unbiased estimate for

$$\nabla F_{q,d}(\mathbf{x}) = \left(\frac{\partial}{\partial x_1} F_{q,d}(\mathbf{x}), \ldots, \frac{\partial}{\partial x_{|D|}} F_{q,d}(\mathbf{x})\right)$$

as follows:

1. Sample one query $q \sim Q$ and one datapoint $d \sim \text{unif}(q(D))$.
2. Build D'': a subset of D that includes each tuple $d_j \in D$ independently with probability $\mathbf{x}_j$.
3. Estimate $\frac{\partial}{\partial x_i} F_{q,d}(\mathbf{x})$ by

$$\begin{cases} 0 & \text{if } d_i \notin q(D), \\ \max\left\{0, \text{sim}(d, d_i) - \max_{d'' \in q(D) \cap D''} \text{sim}(d, d'')\right\} & \text{if } d_i \in q(D). \end{cases} \tag{6}$$

Just like `SCG`, `SCG`^{2} takes T gradient ascent steps in a direction proportional to the largest B entries in the estimated gradient $\nabla F(\mathbf{x})$. After T gradient ascent rounds, the routine yields a solution $\mathbf{x}^*$ in the border of C within a $(1 - 1/e - O(T^{-1/3}))$ approximation factor to the optimal one in expectation. This theoretical guarantee holds due to the fact that `SCG`^{2} inherits its theoretical guarantees from `SCG` as they only differ in the way $\nabla F(\mathbf{x})$ is approximated: From single-stochastic (`SCG`) to doubly-stochastic (`SCG`^{2}). The pseudocode of `SCG`^{2} is provided in Algorithm 2.

Example 5.4. *In our ongoing example (Example 4.1), we ran `SCG`^{2} for $T = 100$ iterations on the convex set $C = \{\mathbf{x} \in [0,1]^3 : \sum_{i=1}^{3} x_i \leq 2\}$ by giving it access to the unbiased estimates of $\nabla F_{q,d}(\mathbf{x})$ as described in Equation 6. Starting from the origin $\mathbf{x}_0 = (0,0,0)$, the algorithm navigates through the continuous space inside the unit cube by making small steps of size $1/T = 1/100$ in the estimated direction of the gradient $\nabla F(\mathbf{x})$. Figure 2 visually depicts one hundred runs of `DepDF` where the orange trajectories correspond to the partial solutions produced by each one of SCG^2 executions. Each such execution returns a continuous vector $\mathbf{x}_T = \mathbf{x}^*$ that lies in the border of C, represented by the diagonal line on the top face of the unit cube depicted in Figure 2. In other words, a continuous vector $\mathbf{x}_T = \mathbf{x}^* = (x_1^*, x_2^*, x_3^*) \in \mathbb{R}^3$ where the first two components, x_1^* and x_2^*, are fractional and satisfy $x_1^* + x_2^* = 1$, while the third component, x_3^*, equals 1.*

Algorithm 2: SCG^2

Data: Multilinear extension F, and budget T.
Result: Continuous solution $\mathbf{x} \in \mathbb{R}^{|D|}$ within a $(1 - 1/e - O(T^{-1/3}))$ approximation factor.

$\mathbf{x}, \mathbf{d} \leftarrow \mathbf{0} \in \mathbb{R}^{|D|}$;
for $t \in [1, \ldots, T]$ **do**
  /* Double-stochastic estimation of ∇F */
  $q \leftarrow q \sim Q$;
  $d \leftarrow d \sim \text{unif}(q(D))$;
  $D'' \leftarrow$ Include $d_j \in D$ with probability $\mathbf{x}_j$;
  $\nabla F_{q,d}(\mathbf{x}) \leftarrow$ eq.(6);
  /* Gradient ascent step as in SCG */
  $\rho_t \leftarrow 1/(2 \cdot (t+1)^{(2/3)})$;
  $\mathbf{d} \leftarrow \mathbf{d} \cdot (1 - \rho_t) + \nabla F_{q,d} * \rho_t$;
  $\mathbf{v} \leftarrow$ Top B coordinates of $\mathbf{d}$;
  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}$
end
$\mathbf{x}^* \leftarrow \mathbf{x}$;
return $\mathbf{x}^*$

Randomized pipage rounding. To bring the continuous solution $\mathbf{x}^* = \mathbf{x}_T$ back into the discrete domain without compromising the approximation factor nor evaluate f, we resort on the *randomized pipage rounding* scheme [3]. Randomized pipage rounding discretizes $\mathbf{x}^*$ into a feasible corner $\mathbf{e}_{D^*}$ of the unit cube that attains a higher utility than $\mathbf{x}^*$, in expectation, without resorting on function evaluation. The final subset of tuples to be retained, $D^* \subseteq D$, is read from the non-zero entries of $\mathbf{e}_{D^*}$.

EXAMPLE 5.5. *In our running example, discretizing* $\mathbf{x}^* = (x_1^*, x_2^*, x_3^*)$*, involves rounding the first two components while maintaining* $x_1^* + x_2^* = 1$*. Naively, we would round* $\mathbf{x}^*$ *to its closest corner along the border of* C *(i.e., along the diagonal line). Looking at Figure 2, we observe that the closest such corner to all produced continuous solutions* $\mathbf{x}^*$ *is corner* $(1, 0, 1)$*, corresponding to subset* $\{d_1, d_3\}$*. However, as mentioned in Example 5.3,* $f(\{d_1, d_3\}) = 0.8974 < F(\mathbf{x}^*) < 0.9306 = f(\{d_2, d_3\})$ *for solutions* $\mathbf{x}^*$*. Randomized pipage rounding allows rounding* $\mathbf{x}^*$ *to the higher utility corner in expectation. In Figure 2,* $\mathbf{x}^*$ *gets rounded to corner* $\mathbf{e}_{D^*} = (0, 1, 1)$ *in expectation. In other words, the highest quality solution* $D^* = \{d_2, d_3\}$ *gets returned in expectation.*

5.1 An Exact Instance

The DepDF algorithm solves Problem 1 with an approximation factor of $(1 - 1/e - \epsilon)$, in expectation, by replacing function evaluation with a double sampling scheme $q \sim Q$, $d \sim \text{unif}(q(D))$, followed by the estimation of $\nabla F_{q,d}(\mathbf{x})$, as described in Equation 6. Interestingly, the problem admits an *exact* solution in a specific scenario. Problem 1 can be exactly solved in polynomial time when the pointwise similarity between datapoints in Function 2 is the step function

$$\text{sim}(d, d') = \begin{cases} 1 & \text{if } d = d', \\ 0 & \text{if } d \neq d'. \end{cases} \tag{7}$$

In this case, Function 2 corresponds to the Jaccard similarity between answer sets $q(D)$ and $q(D')$, which in turn, corresponds to the query precision of q in D' with respect to D. That is, $S(q(D), q(D')) = Jaccard(q(D), q(D')) = Precison(q, D', D)$ for $\text{sim}(\cdot, \cdot)$ defined according to Equation 7.

We refer to Problem 1, where $S(\cdot, \cdot) = Jaccard(\cdot, \cdot)$, as the *independent* data forgetting problem. Coincidentally, this version of the problem matches the setup described in the original vision paper [21], which addresses dataset reduction based on expected query precision maximization. We prove that, under the independence assumption, data forgetting is exactly solvable in polynomial time. In other words, reducing datasets based on expected query precision maximization [21] is exactly solvable in polynomial time. Formally:

THEOREM 5.6. *The optimal solution to the independent data forgetting problem can be obtained using a straightforward approach: Selecting the top B scoring elements in D according to score function* $score(d) = \mathbb{E}_{q \sim Q}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$.

Details of the proof can be found in the extended version of this paper. As a consequence of Theorem 5.6, solving the independent version of the data forgetting problem reduces to a point-wise score computation followed by the extraction of the top B scoring datapoints. This corresponds to the methodology employed by the Independent Data Forgetting (IndepDF) routine (Algorithm 3). Notably, IndepDF eliminates the need for evaluating the objective function f. Instead, it substitutes objective evaluation by $|D|$ independent value assigning operations that are highly parallelizable. Despite its simplicity, IndepDF holds significant theoretical value as it proves that the setting described in the original vision paper [21] is exactly solvable.

Algorithm 3: IndepDF

Data: Dataset D, queryset Q with distribution Q, and budget $B \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ with $|D^*| \leq B$ (exactly) optimizing objective (1) for $S(\cdot, \cdot) = Jaccard(\cdot, \cdot)$.

for $d \in D$ **do**
  $\text{score}(d) \leftarrow \mathbb{E}_{q \sim Q}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$;
end
return Top B points $d \in D$ with highest $\text{score}(d)$;

EXAMPLE 5.7. *We demonstrate the execution of* IndepDF *using our running example.* ***Score computations****: Each tuple* $d \in D$ *is assigned a score. Specifically,* $score(d_1) = \frac{1}{3 \cdot 3} + \frac{1}{3 \cdot 2} = 0.28$, $score(d_2) = \frac{1}{3 \cdot 3} = 0.11$, $score(d_3) = \frac{1}{3 \cdot 3} + \frac{1}{3 \cdot 2} + \frac{1}{3} = 0.61$. ***Output****: The top* 2 *high scoring tuples,* d_1 *and* d_3*, get returned.*

5.2 Complexity Analysis

Table 2: Complexity of our algorithms and that of the state-of-the-art.

Algorithm	Time complexity	Theoretical guarantee
LAZY GREEDY [13, 29]	$O(n \cdot m \cdot B \cdot k^2)$	$(1-1/e)$
LAZY GREEDY + FAST NN [9, 10]	$O(n \cdot m \cdot B \cdot k \cdot b^2)$	$(1-1/e-\epsilon)$
QB-AMNESIA [21]	$O(n \cdot m + B \cdot \log(n))$	None
DepDF (this work)	$O(T \cdot [m + n \cdot \log(n) + n \cdot k])$	$(1-1/e-\epsilon)$
IndepDF (this work)	$O(n \cdot m)$	1 if $sim = (7)$

Let, $n = |D|$, $m = |Q|$, and $k = \max_{q \in Q} |q(D)|$. We start by analyzing the time complexity of our optimization objective $f(D')$, defined in Equation 1. Computing $f(D')$ involves iterating through every query $q \in Q$ and, for each query, calculating $\text{sim}(d, d')$ for every datapoint $d \in q(D)$ and $d' \in q(D')$. Since the queries are monotone $q(D') \subseteq q(D)$, and hence $|q(D')| \leq |q(D)| \leq k$. Consequently, evaluating $f(D')$ requires $O(m \cdot k^2)$ time.

GREEDY [32] is a submodular routine capable of providing a solution to Problem 1 with $(1-1/e)$ approximation guarantee. However, it requires $O(n \cdot B)$ evaluations of the objective function f. Since evaluating f takes $O(m \cdot k^2)$ time, the time complexity of GREEDY is $O(n \cdot m \cdot B \cdot k^2)$.

LAZY GREEDY, originally introduced by Minoux [29] and used for forgetting e-commerce data by Gershtein et al. [13], is a submodular routine capable of providing a solution to Problem 1 with $(1-1/e)$ approximation guarantee. It exploits lazy evaluation to speed up GREEDY. Although lazy evaluation enable LAZY GREEDY to run orders of magnitude faster than GREEDY in practical scenarios, both manifest the same computational complexity. In fact, it is unknown the number of function evaluations performed by LAZY GREEDY in arbitrary cases, but it is certain that it matches that of GREEDY in the worst one. Therefore, the time complexity of LAZY GREEDY is $O(n \cdot m \cdot B \cdot k^2)$.

LAZY GREEDY + FAST NN, recently proposed by Davidson et al. [9, 10] for forgetting image data, exploits fast approximate nearest-neighbor computations to speed up function evaluation in LAZY GREEDY. More precisely, the algorithm exploits LSH to only consider similarities between vector pairs corresponding to hash collisions. Hence, if b denotes the size of the biggest bucket, evaluating the data forgetting objective $f(D')$ requires $O(m \cdot k \cdot b^2)$ time as opposed to $O(m \cdot k^2)$ time. This will be beneficial if $b^2 << k$. Overall, the time complexity of LAZY GREEDY + FAST NN is $O(n \cdot m \cdot B \cdot k \cdot b^2)$.

QUERY-BASED-AMNESIA (QB-AMNESIA) [21] is a sampling algorithm that randomly selects B tuples from D with a probability proportional to their retrieval frequency according to distribution Q. The algorithm does not require evaluating the function $f(D')$ to construct a solution. However, it provides no theoretical guarantees about the quality of the solution. Time-complexity-wise, computing the tuple frequency requires $O(n \cdot m)$ time and sampling B points from D based on the computed frequencies requires $O(n + B \cdot \log(n))$ time. This is because calculating the cumulative distribution function (CDF) from the given vector of frequency probabilities takes $O(n)$ time and each independently drawn sample takes $O(\log(n))$ due to binary search of the CDF. Hence, the time complexity of QUERY-BASED-AMNESIA is $O(n \cdot m + B \cdot \log(n))$.

DepDF, the first algorithm proposed in this work, is a doubly-stochastic submodular routine that yields a solution with $(1-1/e-O(T^{-1/3}))$ approximation guarantee in expectation. The algorithm does not require evaluating objective function $f(D')$ to construct such solution. Instead, it relies on executing SCG^2, a double-stochastic extension of SCG, on the multilinear extension of f, denoted F. This translates to T rounds of estimating gradient ∇F. SCG^2 achieves this by: sampling one query $q \sim Q$ ($O(m + \log(m))$ time), sampling one datapoint $d \sim \text{unif}(q(D))$ ($O(1)$ time as the distribution is uniform), and estimating $\nabla F_{q,d}(\mathbf{x})$ (i.e, n partial derivatives $\frac{\partial}{\partial x_i} F_{q,d}(\mathbf{x})$). This requires building D'', a subset of D that includes each tuple $d_j \in D$ independently with probability $\mathbf{x}_j$ ($O(n + n \cdot \log(n))$ time) and n evaluations of Equation 6 ($O(n \cdot k)$ time). After $\nabla F_{q,d}$ is estimated, a gradient ascent step is taken in the direction proportional of the biggest B components. Finding indices of the top B components can be done in $O(n)$ time by partial sorting. The $B-$th element will be in its final sorted position and all smaller elements are moved before it and all larger elements behind it. Last, randomized pipage rounding takes $O(n)$ time in our problem-specific setting. Therefore, the time complexity of DepDF is $O(T \cdot [m + n \cdot \log(n) + n \cdot k])$.

IndepDF, the second algorithm we propose in this work, is a deterministic routine capable of providing an exact solution to Problem 1 in the case where the point-wise similarity measure $\text{sim}(\cdot, \cdot)$ is defined according to Equation 7. Nevertheless, it provides no theoretical guarantees for any other similarity measure. The algorithm performs a point-wise score-assigning operation described in Theorem 5.6 that takes $O(n \cdot m)$ time. Subsequently, the top B scoring elements get returned. This is achieved in $O(n)$ time by the use of partial sorting. Hence, the time complexity of IndepDF is $O(n \cdot m)$. Nonetheless, it is important to remark that the point-wise score-assigning operations performed by IndepDF are independent, and hence are highly parallelizable.

6 Experimental evaluation

We performed an extensive evaluation to experimentally prove the effectiveness of our proposed algorithms. More specifically, we conducted experiments to answer: (i) Can DepDF or IndepDF produce solutions of comparable quality to those produced by the state-of-the-art submodular algorithms while exhibiting a runtime comparable to that of the state-of-the-art amnesia algorithms? (ii) Despite not having general theoretical guarantees, can IndepDF produce high-quality solutions under specific data-dependent conditions? and (iii) Can DepDF and IndepDF forget datasets at scale? In what follows, we explain how the experiments were performed and report the findings with respect to the above three questions. Our code is available at " https://github.com/rricocuevas1/ssdf ".

6.1 Experimental Setup

We tested our proposed algorithms against three state-of-the-art data forgetting routines on eight datasets using a server with 1 TB RAM

Table 3: Characteristics of the considered datasets.

Name	Size: $\|D\|$	Query-log size: $\|Q\|$	Nature
Flights	1.200	37	categorical
Photos	41.620	443	numerical
Wiki	131.148	14.723	numerical
$S_{(1M,\,100K)}$	1.000.000	100.000	numerical
$S_{(1M,\,1M)}$	1.000.000	1.000.000	numerical
$S_{(1M,\,10M)}$	1.000.000	10.000.000	numerical
$S_{(5M,\,10M)}$	5.000.000	10.000.000	numerical
$S_{(10M,\,10M)}$	10.000.000	10.000.000	numerical

and 112 cores. We let each algorithm run for a maximum of three days in each of the experiments that we performed.

[Baselines]. We compare the `IndepDF` and `DepDF` algorithms against three state-of-the-art data forgetting methods:
(i) `QB-AMNESIA` [21], which is the leading approach in amnesia-based data forgetting; (ii) `LAZY GREEDY` [13, 29], which is the state-of-the-art submodular-based data forgetting method; and (iii) `LAZY GREEDY + FAST NN` [9, 10], a top accelerated submodular-based data forgetting technique.

[Datasets]. We used three real datasets coming from different sources with corresponding real query-logs, and five synthetic datasets with synthetic query-logs. A summary of their characteristics is displayed in Table 3 and are freely available for download[3]. The datasets, specifically, are:

- **[Flights]** We considered the `flights.csv` file in folder `flight_2` of the spider dataset.[4] Its query-log corresponds to all of the queries that refer to the `flights.csv` table in files `dev_gold.sql` and `train_gold.sql`, also from the spider dataset. The queries adhere to the hypothesis in the paper, so no filtering was required.
- **[Photos]** We selected the validation set of a publicly available dataset of images [23]. Each image is assigned a set of labels that describe the objects that are present in it. Naturally, each label can be understood as a select query. Namely, one that returns all images tagged with such label. In order to have balanced-size answer sets, only those labels appearing in at most 100 images were kept. Further, in order to fit our assumed data model, images were embedded using the pretrained ResNet-50 network [15]. Hence, every image is identified with a vector in $\mathbb{R}^{2048}$.
- **[Wikidata]** We chose the portion of Wikidata[5] retrieved by the SPARQL queries executed by users via the Wikidata Query Service[6] between 12/06/2017 and 09/07/2017[7]. To have balanced-size answer sets, we only kept those queries whose answer sets were smaller than 100. Due to the nature of the data, some pre-processing was required. Every webpage is identified by a unique id, and any two webpages only share the "label" and "description" fields. Hence, given any id, we concatenated its corresponding webpage label and description texts into a single sentence and embedded the latter using the "all-MiniLM-L6-v2" sentence transformer. This way, every webpage is identified with a vector in $\mathbb{R}^{384}$. The queries adhere to our hypothesis, so no filtering was required. We only excluded those that returned empty answers.
- **[Synthetic]** We created five synthetic datasets containing up to 10.000.000 tuples coupled with synthetic query-logs containing up to 10.000.000 queries. Each tuple contains 100 numerical values and each query retrieves a fixed number of unique tuples. We opted for this simplified workload as we are only interested in studying the scalability of the algorithms. Their ability to cope with real-world workloads is already tested with the three aforementioned real datasets. We denote by $S_{(n,\,m)}$ a dataset with n synthetic tuples coupled with a query-log with m synthetic queries. The characteristics of our five synthetically generated datasets are displayed in Table 3.

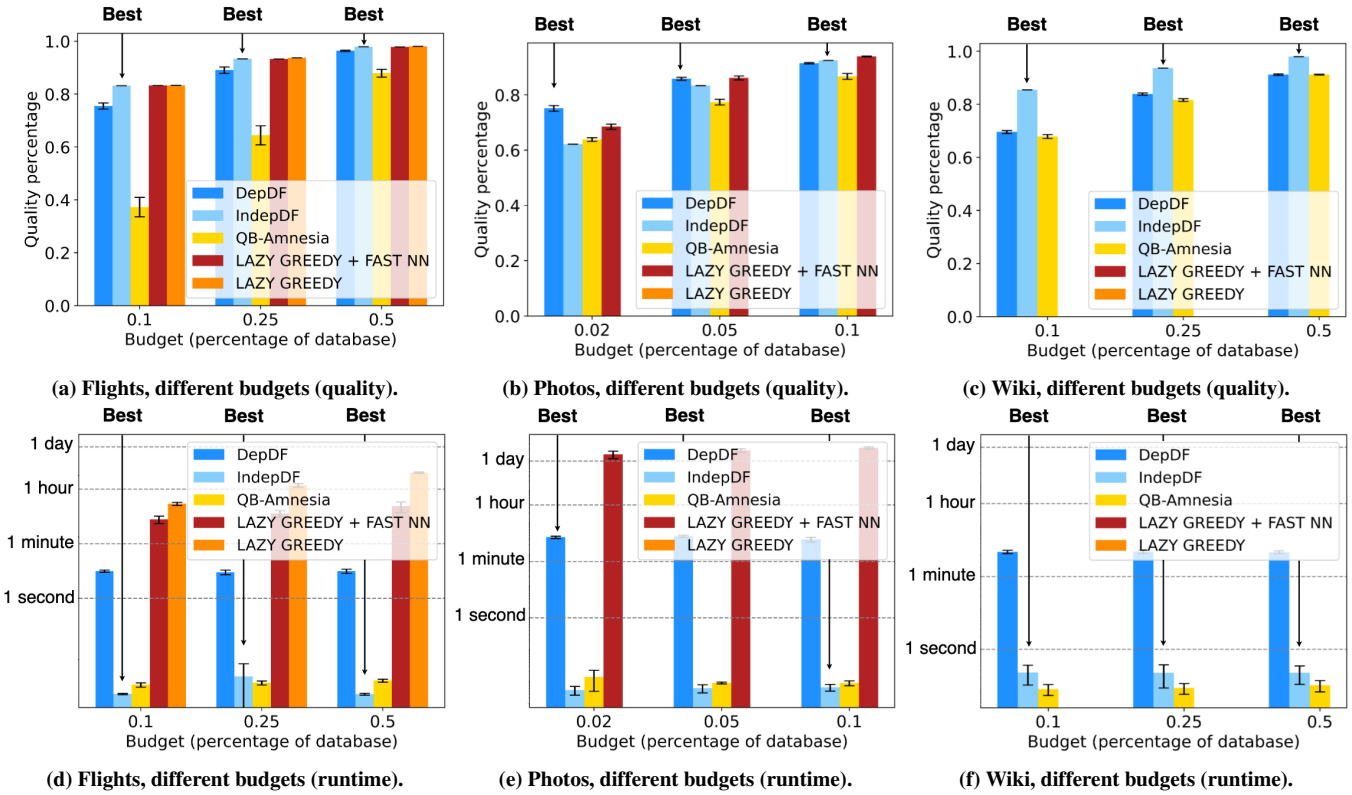
[Metrics]. To measure the quality of the produced solution we used the data forgetting optimization objective $f(D')$, $D' \subseteq D$, defined in Equation 1. The values of f range between 0 and 1 with $f(D) = 1$ (i.e., not deleting anything preserves 100% of the information). To measure the runtime performance, we measured the execution time.

6.2 Quality vs Runtime

Figure 3 portrays the quality and runtime performance of our algorithms against that of the baselines when forgetting three real-world datasets with different budgets. As illustrated in the figure, our proposed data forgetting routines strike the best balance between quality and runtime. More precisely, either `IndepDF` or `DepDF` *always* outperforms or matches quality-wise `LAZY GREEDY` and `LAZY GREEDY + FAST NN` (the state-of-the-art submodular algorithms), while exhibiting a runtime comparable to that of the `QB-AMNESIA` routine (the state-of-the-art amnesia algorithm).

Particularly, with respect to the Flights dataset (Figures 3a and 3d): `IndepDF` produced solutions of indistinguishable quality to those generated by `LAZY GREEDY` and `LAZY GREEDY + FAST NN` across all budgets, while presenting a runtime equal or inferior to that of `QB-AMNESIA`. Regarding the Photos dataset (Figures 3b and 3e), for the smallest budget configurations, `DepDF` generated the best quality solution in the order of minutes; and for the largest budget configuration, `IndepDF` retrieved the best quality solution in less than a second. We highlight that this runtime performance is much lower than that of the next best performing algorithm (`LAZY GREEDY + FAST NN`), which took more than one day to fetch a quality-equivalent solution. Furthermore, `LAZY GREEDY` did not complete any of the experiments over the course of three days. Last, regarding the Wiki dataset (Figures 3c and 3f): `IndepDF` produced the best quality solution across every budget configuration, with a runtime equivalent to that of `QB-AMNESIA`. In this context, neither `LAZY GREEDY` nor `LAZY GREEDY + FAST NN` were able to complete any experiment over the course of three days, while again, `IndepDF` obtained a solution in less than one second.

[3] https://drive.google.com/file/d/1YjCt-RZUyEHslqmA3yJHJi-Tk6SNFlbP/view
[4] https://drive.google.com/uc?export=download&id=1TqleXec_OykOYFREKKtschzY29dUcVAQ
[5] https://www.wikidata.org/wiki/Wikidata:Main_Page
[6] https://query.wikidata.org/
[7] https://analytics.wikimedia.org/datasets/one-off/wikidata/sparql_query_logs/2017-06-12_2017-07-09/2017-06-12_2017-07-09_organic.tsv.gz



(a) Flights, different budgets (quality). (b) Photos, different budgets (quality). (c) Wiki, different budgets (quality).

(d) Flights, different budgets (runtime). (e) Photos, different budgets (runtime). (f) Wiki, different budgets (runtime).

Figure 3: Quality and runtime performance of each algorithm when forgetting real-world datasets with different budget configurations. The arrows starting with "Best" point to the algorithm that best balances quality and runtime in each budget configuration.

6.3 Quality guarantees of the DepDF and IndepDF

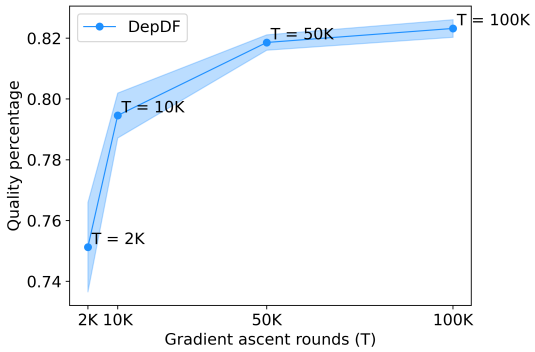
DepDF enjoys very strong theoretical guarantees. It yields solutions with $1 - 1/e - O(T^{-1/3})$ approximation guarantee to the optimal one, in expectation. That is, theoretically, as the number of gradient ascent rounds T grows, the quality of the produced solution grows showing diminishing returns. Indeed, as can be seen in Figure 4, this behavior is also manifested experimentally.

Theoretically, IndepDF only enjoys quality guarantees when the point-wise similarity measure $\text{sim}(\cdot,\cdot)$ is defined according to Equation 7. Nevertheless, as can be seen in Figure 3, IndepDF enjoys extremely high quality-performance in the Flights and Wiki datasets where $\text{sim}(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in Equation 3, and $\text{sim}(\cdot,\cdot) = cos(\cdot,\cdot)$ as in Equation 4, respectively.

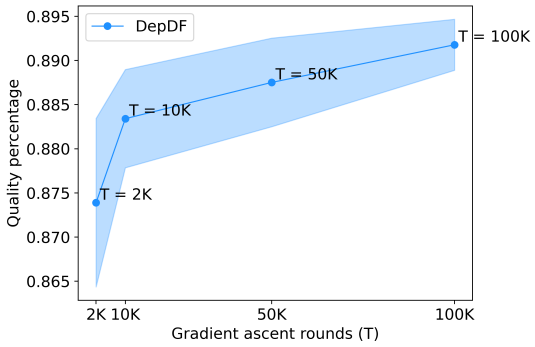
Despite not having theoretical guarantees for arbitrary similarity functions, IndepDF enjoys strong data-dependent guarantees based on answer set diversity. The average answer set diversity of a dataset D over a query log Q is the average pointwise similarity $\text{sim}(d, d')$ between every $d, d' \in q(D)$ for every $q \in Q$. The average answer set diversity of the Flights, and Wiki datasets over their corresponding query-logs is 0.057, and 0.037, respectively. That is, the answer sets have very low diversity on average (i.e., they are homogeneous on average). Intuitively, if answer sets are homogeneous, each datapoint within each answer set is (essentially) of the same importance to the latter. Hence, *any* point-wise similarity measure (particularly, the Jaccard and cosine similarity) converges to Equation 7. Consequently, the quality of the produced solutions by IndepDF should, in theory, approach that of the state-of-the-art submodular algorithms as the diversity of answer sets decrease. A depiction of this convergence is portrayed in Figure 5 for the Jaccard and the cosine similarity (Figures 5a and 5b, respectively). Therefore, despite not having general theoretical guarantees, IndepDF is a quality-effective approach for data forgetting when the query-log retrieves homogeneous views.

6.4 Scalability of the DepDF and IndepDF algorithms

As illustrated in Figure 3, function evaluation acceleration (i.e, the algorithmic approach taken by LAZY GREEDY and LAZY GREEDY + FAST NN) is not a feasible solution towards scalable data forgetting. More specifically, as shown in Figures 3b and 3e, LAZY GREEDY was not able to complete any experiment on the Photos dataset (i.e., a small-sized dataset) over the course of three days. Further, looking at figures 3c and 3f, we note that neither LAZY GREEDY nor LAZY GREEDY + FAST NN were capable of completing any experiment on the Wiki dataset (i.e., a mid-sized dataset) over the course of three days. This clearly showcases that the state-of-the-art submodular-based routines with theoretical guarantees that rely on function evaluation acceleration, do not allow data forgetting beyond scale mid-sized datasets. Consequently, the state-of-the-art

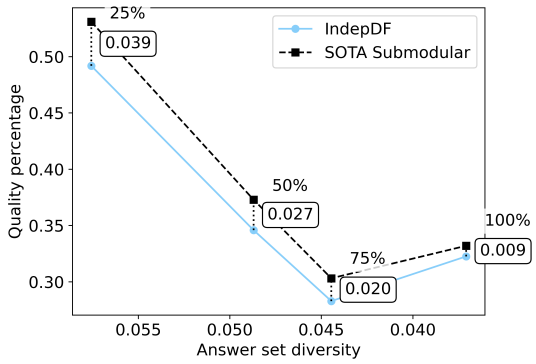


(a) Flights, increasing T (quality).

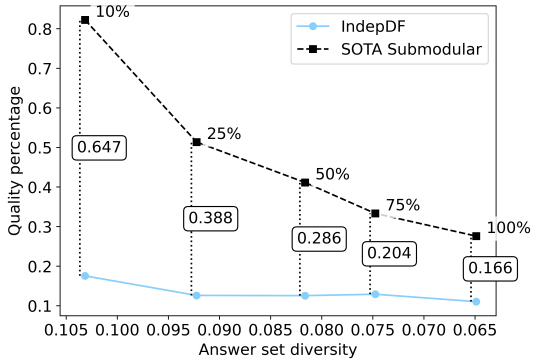


(b) Photos, increasing T (quality).

Figure 4: Quality performance of `DepDF` when increasing the number gradient ascent rounds T.



(a) Flights, decreasing $q(D)$-diversity (quality).



(b) Photos, decreasing $q(D)$-diversity (quality).

Figure 5: Quality performance difference between the best performing state-of-the-art submodular algorithm and `IndepDF` when decreasing answer set diversity. $n\%$ indicates the top n percent most diverse queries in the query-log.

submodular-based approaches are incapable of performing quality-preserving data forgetting at scale.

Furthermore, as can be seen in Figure 3, despite being able to produce solutions rapidly `QB-AMNESIA` yields the worst-quality solution across every dataset and budget configuration. Consequently, the state-of-the-art amnesia-based approach is incapable of performing quality-preserving data forgetting at scale.

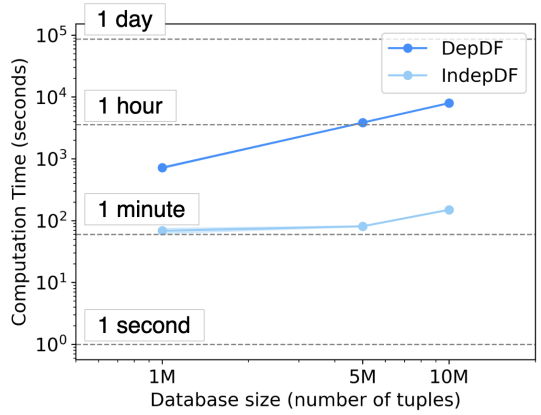
In contrast, as can be seen in Figures 6a and 6b, `DepDF` and `IndepDF` are able to scale to datasets up to two orders of magnitude larger in database size and three orders of magnitude larger in query log size relative to existing submodular algorithms with which they share the same quality performance. Hence, the `DepDF` and `IndepDF` algorithms are an effective solution for performing quality-preserving data forgetting at scale.

7 Conclusion

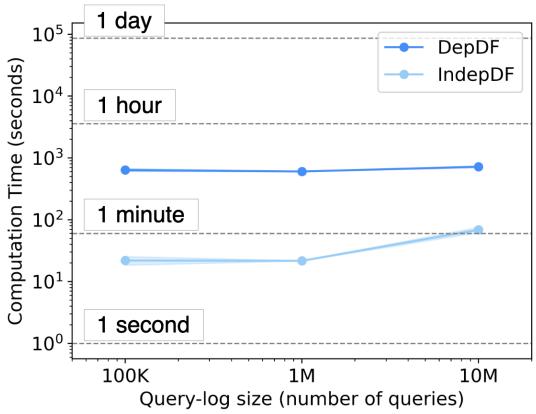
In this work we address the data reduction challenge that many organizations are inevitably facing. We propose what is, to the best of our knowledge, the first stochastic and topology-aware formulation for the new data reduction paradigm where data importance is understood as data usage. Our formulation unifies the ideas presented in [8–10, 13, 21] under the same formal framework, which we suggest to be called **data forgetting**. Furthermore, we propose two data forgetting routines, `DepDF` and `IndepDF`, that effectively bridge the gap between the limitations of state of the art submodular-based routines and amnesia algorithms. Our proposed methods enjoy scalability due to function evaluation avoidance, while maintaining strong theoretical quality guarantees. Specifically, `DepDF` enjoys strong theoretical guarantees for the general data forgetting problem while `IndepDF` only does so, theoretically, when $\text{sim}(\cdot,\cdot)$ is defined according to Equation 7. In spite of this, our experiments show that the `IndepDF` algorithm exhibits strong data-dependent guarantees based on answer set diversity.

References

[1] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. 2000. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, Vol. 2000. 496–505.
[2] Mohiuddin Ahmed. 2019. Data summarization: a survey. *Knowledge and Information Systems* 58 (2019), 249–273.
[3] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766.
[4] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 511–519.
[5] William G. Cochran. 1977. *Sampling Techniques, 3rd Edition.* John Wiley.
[6] Graham Cormode. 2017. Data Sketching. *Commun. ACM* 60, 9 (aug 2017), 48–55.



(a) Synthetic data, increasing $|D|$ (runtime).



(b) Synthetic data, increasing $|Q|$ (runtime).

Figure 6: Scalability study of `DepDF` and `IndepDF`.

[7] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.

[8] Susan B. Davidson, Shay Gershtein, Tova Milo, and Slava Novgorodov. 2022. Disposal by Design. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2022).

[9] Susan B. Davidson, Shay Gershtein, Tova Milo, Slava Novgorodov, and May Shoshan. 2022. PHOcus: Efficiently Archiving Photos. *Proc. VLDB Endow.* 15, 12 (sep 2022), 3630–3633.

[10] Susan B Davidson, Shay Gershtein, Tova Milo, Slava Novgorodov, and May Shoshan. 2023. Efficiently Archiving Photos under Storage Constraints. *EDBT* (2023).

[11] Jonas Fischer and Jilles Vreeken. 2021. Differentiable pattern set mining. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 383–392.

[12] Minos N Garofalakis and Phillip B Gibbons. 2001. Approximate Query Processing: Taming the TeraBytes.. In *VLDB*, Vol. 10. 645927–672356.

[13] Shay Gershtein, Tova Milo, and Slava Novgorodov. 2020. Inventory Reduction via Maximal Coverage in E-Commerce.. In *EDBT*. 522–533.

[14] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. 2017. Gradient methods for submodular maximization. *Advances in Neural Information Processing Systems* 30 (2017).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Holon Investments. 2023. Global data storage and manufacturing capacity is critically low. *The Holon Data Report, Part 5* (2023).

[17] Mohammad Karimi, Mario Lucic, Hamed Hassani, and Andreas Krause. 2017. Stochastic submodular maximization: The case of coverage functions. *Advances in Neural Information Processing Systems* 30 (2017).

[18] Howard Karloff and Milena Mihail. 1999. On the complexity of the view-selection problem. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 167–173.

[19] Martin Kersten. 2015. Big Data Space Fungus. In *Proceedings of the 7th CIDR*.

[20] Martin Kersten. 2016. Keynote: DataFungi, from Rotting Data to Purified Information.. In *Proceedings of the 32nd ICDE*.

[21] Martin Kersten and Lefteris Sidirourgos. 2017. A Database System with Amnesia. In *CIDR*.

[22] Andreas Krause and Daniel Golovin. 2014. Submodular function maximization. *Tractability* 3 (2014), 71–104.

[23] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV* (2020).

[24] Kaiyu Li and Guoliang Li. 2018. Approximate query processing: What is new and where to go? a survey on approximate query processing. *Data Science and Engineering* 3, 4 (2018), 379–397.

[25] Hui Lin and Jeff A Bilmes. 2012. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871* (2012).

[26] Qing Liu. 2009. *Approximate Query Processing*. Springer US, Boston, MA, 113–119. https://doi.org/10.1007/978-0-387-39940-9_534

[27] Imene Mami and Zohra Bellahsene. 2012. A survey of view selection methods. *Acm Sigmod Record* 41, 1 (2012), 20–29.

[28] Tova Milo. 2019. Getting Rid of Data. *J. Data and Information Quality* 12, 1, Article 1 (nov 2019), 7 pages.

[29] Michel Minoux. 1978. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1978*. Springer, 234–243.

[30] Hoshi Mistry, Prasan Roy, S Sudarshan, and Krithi Ramamritham. 2001. Materialized view selection and maintenance using multi-query optimization. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 307–318.

[31] Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. 2018. Conditional gradient method for stochastic submodular maximization: Closing the gap. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1886–1895.

[32] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14 (1978), 265–294.

[33] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.

[34] Jeff M Phillips. 2017. Coresets and sketches. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1269–1288.

[35] David Reinsel, John Gantz, and John Rydning. 2018. Data Age 2025 - The digitization of the world from edge to core. *Seagate* (2018).

[36] Amit Shukla, Prasad Deshpande, Jeffrey F Naughton, et al. 1998. Materialized view selection for multidimensional datasets. In *VLDB*, Vol. 98. 488–499.

[37] Ian Simon, Noah Snavely, and Steven M Seitz. 2007. Scene summarization for online image collections. In *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 1–8.

[38] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. 2014. Learning Mixtures of Submodular Functions for Image Collection Summarization. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc.

[39] Graham J. G. Upton and Patrick Cook. 2002. A Dictionary of Statistics.

[40] TV Vijay Kumar and Santosh Kumar. 2012. Materialized view selection using genetic algorithm. In *Contemporary Computing: 5th International Conference, IC3 2012, Noida, India, August 6-8, 2012. Proceedings 5*. Springer, 225–237.

[41] Jan Vondrák. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 67–74.

.

A Appendix

A.1 Proof of Theorem 5.1

For notational convenience, we will denote $f_{q,d}(D') := \max_{d' \in q(D')} \text{sim}(d, d')$.

PROOF.

1. Normalized non-negativity: This is an immediate consequence of the fact that $\text{sim}(d, d') \in [0, 1]\ \forall\, d, d' \in D$.

2. Monotonicity: $A \subseteq B \subseteq D \implies q(A) \subseteq q(B) \subseteq q(D) \implies f_{q,d}(A) = \max_{d' \in q(A)} \text{sim}(d, d') \leq \max_{d' \in q(B)} \text{sim}(d, d') = f_{q,d}(B) \implies f(A) \leq f(B)$. The last implication follows from the fact that sums and expectations respect monotonicity.

3. Submodularity: Given any $A \subseteq B \subseteq D$ and $d^* \in D \setminus B$, we distinguish two cases:

- **Case 1:** $q(\{d^*\}) = \emptyset$. In this scenario,

$$\begin{aligned}\Delta_{f_{q,d}}(d^*|B) &= \max_{d' \in q(B \cup \{d^*\})} \text{sim}(d, d') - \max_{d' \in q(B)} \text{sim}(d, d') \\ &\underset{q(B \cup \{d^*\}) = q(B) \cup q(\{d^*\}) = q(B)}{=} 0 \\ &\underset{q(A \cup \{d^*\}) = q(A) \cup q(\{d^*\}) = q(A)}{=} \Delta_{f_{q,d}}(d^*|A).\end{aligned}$$

- **Case 2:** $q(\{d^*\}) = d^*$. In this scenario,

$$\begin{aligned}\Delta_{f_{q,d}}(d^*|B) &= \max_{d' \in q(B \cup \{d^*\})} \text{sim}(d, d') - \max_{d' \in q(B)} \text{sim}(d, d') \\ &= \max\left\{0, \text{sim}(d, d^*) - \max_{d' \in q(B)} \text{sim}(d, d^*)\right\}.\end{aligned}$$

Now, for any $d \in D$,

$$\begin{aligned}A \subseteq B &\implies q(A) \subseteq q(B) \\ &\implies \max_{d' \in q(A)} \text{sim}(d, d') \leq \max_{d' \in q(B)} \text{sim}(d, d') \\ &\implies -\max_{d' \in q(B)} \text{sim}(d, d') \leq -\max_{d' \in q(A)} \text{sim}(d, d') \\ &\implies \text{sim}(d, d^*) - \max_{d' \in q(B)} \text{sim}(d, d') \\ &\quad \leq \text{sim}(d, d^*) - \max_{d' \in q(A)} \text{sim}(d, d').\end{aligned}$$

Hence, $\Delta_{f_{q,d}}(d^*|B) \leq \Delta_{f_{q,d}}(d^*|A)$.

This proves that functions $f_{q,d}(D')$ are submodular. The submodularity of $f(D')$ follows from the fact that both the sum and expectation operators preserve submodularity [22]. □

A.2 Proof of Theorem 5.2

LEMMA A.1. *For any given datasets $D' \subseteq D$ and monotone SPJU query q,*

$$|q(D')| = \sum_{d \in D'} |q(\{d\})|.$$

PROOF. $D' = \cup_{d \in D'} \{d\} \underset{(1.1)}{\implies} q(D') = q(\cup_{d \in D'} \{d\}) \underset{(1.2)}{\implies} q(D') = \cup_{d \in D'} q(\{d\}) \underset{(1.3)}{\implies} |q(D')| = |\cup_{d \in D'} q(\{d\})| \underset{(1.4)}{\implies} |q(D')| = \sum_{d \in D'} |q(\{d\})|$. ($\cup$ denotes the disjoint union). (1.1) holds because the answer set of any query q on any subset $D' \subseteq D$ is unique. (1.2) holds because q distributes under set union. (1.3) holds because equal sets have the same cardinality. (1.4) holds because of the inclusion-exclusion principle in the case of a disjoint union. □

To prove that the objective is double-stochastic we must prove that there exists a probability distribution $\mathcal{D}$ such that $f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, f_{q,d}(D')$.

$$
\begin{aligned}
f(D') &= \mathbb{E}_{q\sim Q}[S(q(D), q(D'))] \\
&= \mathbb{E}_{q\sim Q}\left[\frac{1}{|q(D)|}\cdot\sum_{d\in q(D)}\max_{d'\in q(D')}\mathrm{sim}(d,d')\right] \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\cdot\frac{1}{|q(D)|}\sum_{d\in q(D)} f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\sum_{d\in q(D)}\frac{1}{|q(D)|}\cdot f_{q,d}(D') \qquad (*) \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\left(\sum_{d\in q(D)}\frac{1}{|q(D)|}\cdot f_{q,d}(D') + \sum_{d\in D\setminus q(D)} 0\right) \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\sum_{d\in D}\frac{\mathbb{I}_{\{d\in q(D)\}}(d)}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\sum_{d\in D}\frac{|q(\{d\})|}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_Q(q)\cdot\frac{|q(\{d\})|}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_{\mathcal{D}}(d,q)\cdot f_{q,d}(D') \\
&= \sum_{d\in D}\sum_{q\in Q}\mathbb{P}_{\mathcal{D}}(d,q)\cdot f_{q,d}(D') \\
&= \mathbb{E}_{(d,q)\sim\mathcal{D}}\,[f_{q,d}(D')].
\end{aligned}
$$

Hence, $f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, f_{q,d}(D')$ for $\mathcal{D}$ with probability mass function $\mathbb{P}_{\mathcal{D}}(d,q) = \frac{|q(\{d\})|}{|q(D)|}\cdot\mathbb{P}_Q(q), \quad \forall\,(d,q)\in D\times Q$. Last, we show this is indeed a probability mass function:

1. Normalized non-negativity: $\mathbb{P}_{\mathcal{D}}(d,q)\geq 0$: This is true since $\frac{|q(\{d\})|}{|q(D)|}\geq 0$ and $\mathbb{P}_Q(q)\in[0,1]$. Note that queries are assumed to be non-empty (i.e., $q(D)\neq\emptyset$, so no division by 0 takes place).

2. Sum up to 1: $\sum_{(d,q)\in D\times Q}\mathbb{P}_{\mathcal{D}}(d,q) = 1$:

$$
\begin{aligned}
&\sum_{(d,q)\in D\times Q}\mathbb{P}_{\mathcal{D}}(d,q) = \sum_{d\in D}\sum_{q\in Q}\mathbb{P}_{\mathcal{D}}(d,q) = \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_{\mathcal{D}}(d,q) \\
&= \sum_{q\in Q}\sum_{d\in D}\frac{|q(\{d\})|}{|q(D)|}\cdot\mathbb{P}_Q(q) = \sum_{q\in Q}\mathbb{P}_Q(q)\sum_{d\in D}\frac{|q(\{d\})|}{|q(D)|} \\
&= \sum_{q\in Q}\mathbb{P}_Q(q)\frac{\sum_{d\in D}|q(\{d\})|}{|q(D)|} \underset{\text{Lemma A.1}}{=} \sum_{q\in Q}\mathbb{P}_Q(q)\frac{|q(D)|}{|q(D)|} = 1.
\end{aligned}
$$

Next, we prove the second part of the theorem, i.e., the fact that estimating $\nabla F_{q,d}(\mathbf{x})$ is less costly than estimating $\nabla F_q(\mathbf{x})$. Recall that $F_{q,d}(\mathbf{x})$ denotes the multilinear extension of $f_{q,d}(D') = \max_{d'\in q(D')}\mathrm{sim}(d,d')$, and $F_q(\mathbf{x})$ corresponds to the multilinear extension of $f_q(D') = S(q(D), q(D'))$ with $S(\cdot,\cdot)$= (2). The equation marked as (*) implies that

$$f(D') = \mathbb{E}_{q\sim Q}\mathbb{E}_{d\sim\mathrm{unif}(q(D))}[f_{q,d}(D')].$$

Therefore, $f_q(D') = \mathbb{E}_{d\sim\mathrm{unif}(q(D))}[f_{q,d}(D')]$. Further, by the linearity of the multilinear extension [41], $F_q(\mathbf{x}) = \mathbb{E}_{d\sim\mathrm{unif}(q(D))}[F_{q,d}(\mathbf{x})]$. Since ∇ is linear, $\nabla F_q(\mathbf{x}) = \mathbb{E}_{d\sim\mathrm{unif}(q(D))}[\nabla F_{q,d}(\mathbf{x})]$. Given that queries are assumed to be non-empty, $|q(D)|\geq 1$. Thus, estimating $\nabla F_q(\mathbf{x})$ requires estimating at least as many terms as estimating $\nabla F_{q,d}(\mathbf{x})$.

A.3 Proof of Theorem 5.6

If $S(q(D), q(D')) = Jaccard(q(D), q(D')) = Precison\ (q, D', D)$, objective 1 becomes finding a subset $D^*\subseteq D$ such that

$$
\begin{aligned}
D^* &= \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \mathbb{E}_{q\sim Q}\left[\frac{|q(D')|}{|q(D)|}\right] \\
&= \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \sum_{q\in Q}\mathbb{P}_Q(q)\cdot\frac{|q(D')|}{|q(D)|} \\
&\underset{\text{Lemma A.1}}{=} \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \sum_{q\in Q}\mathbb{P}_Q(q)\cdot\frac{\sum_{d\in D'}|q(\{d\})|}{|q(D)|} \\
&= \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \sum_{q\in Q}\mathbb{P}_Q(q)\cdot\sum_{d\in D'}\frac{|q(\{d\})|}{|q(D)|} \\
&= \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \sum_{q\in Q}\sum_{d\in D'}\mathbb{P}_Q(q)\cdot\frac{|q(\{d\})|}{|q(D)|} \\
&= \underset{D'\subseteq D,\ |D'|\leq B}{\arg\max}\ \sum_{d\in D'}\sum_{q\in Q}\mathbb{P}_Q(q)\cdot\frac{|q(\{d\})|}{|q(D)|} \\
&= \underset{D'\subseteq D,\ \sum_{d\in D'}1\leq B}{\arg\max}\ \sum_{d\in D'}\mathbb{E}_{q\sim Q}\left[\frac{|q(\{d\})|}{|q(D)|}\right].
\end{aligned}
$$

Hence, the optimal solution to the independent data forgetting problem corresponds to those B elements in D that have the highest score according to $\mathrm{score}(d) = \mathbb{E}_{q\sim Q}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$.

This result extends beyond the simple cardinality constraint. In particular, it holds for the (more general) Knapsack constraint. If the cost of each tuple $d\in D$ is given by a cost function $C: D\to\mathbb{R}_+$ where $C(D') := \sum_{d\in D'}C(d)\ \forall D'\subseteq D$, the problem reduces to the 0-1 Knapsack problem where the set of n items corresponds to the dataset D, $v_d = \mathbb{E}_{q\sim Q}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$, $w_d = C(d)$, and $W = B$

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009