Explore or Exploit? Effective Strategies for Disambiguating Large Databases

Reynold Cheng[†] Eric Lo[‡] Xuan S. Yang[†] Ming-Hay Luk[‡] Xiang Li[†] Xike Xie[†]
[†] University of Hong Kong, Pokfulam Road, Hong Kong
[‡] Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[†] {ckcheng, xyang2, xli, xkxie}@cs.hku.hk [‡]{ericlo, csmhluk}@comp.polyu.edu.hk

ABSTRACT

Data ambiguity is inherent in applications such as data integration, location-based services, and sensor monitoring. In many situations, it is possible to "clean", or remove, ambiguities from these databases. For example, the GPS location of a user is inexact due to measurement errors, but context information (e.g., what a user is doing) can be used to reduce the imprecision of the location value. In order to obtain a database with a higher quality, we study how to disambiguate a database by appropriately selecting candidates to clean. This problem is challenging because cleaning involves a cost, is limited by a budget, may fail, and may not remove all ambiguities. Moreover, the statistical information about how likely database objects can be cleaned may not be precisely known. We tackle these challenges by proposing two types of algorithms. The first type makes use of greedy heuristics to make sensible decisions; however, these algorithms do not make use of cleaning information and require user input for parameters to achieve high cleaning effectiveness. We propose the *Explore-Exploit* (or *EE*) algorithm, which gathers valuable information during the cleaning process to determine how the remaining cleaning budget should be invested.

We also study how to fine-tune the parameters of *EE* in order to achieve optimal cleaning effectiveness. Experimental evaluations on real and synthetic datasets validate the effectiveness and efficiency of our approaches.

1. INTRODUCTION

In many applications, data is often inexact or imprecise. For example, schema matching tools such as COMA [11] can be used to perform integration over the schemas of some flight-ticket agents' websites. Since these matching tools typically associate an attribute with more than one attribute from other schemas, a set of possible prices can be associated with each flight. As another example, consider a database that captures customers' movie ratings obtained by the fusion of IMDB movie information and ratings obtained from the Netflix challenge [23]. In this database, a user can give a rating to the same movie at different times. She may also input a different name for the same movie (e.g., "Disney's Snow White" and "Snow White"). Additionally, the record linkage tools used for integrating the databases are not perfect. The customer's rating for each movie in the database is then not precise and appears as a set of possible values instead. In applications that manage crime-solving-related information, evidence data provided by witnesses (e.g., hair color, brand of the car seen), as well as identities of true offenders, are ambiguous [22]. In Global-Positioning Systems (GPS) [19], RFID systems, and habitat monitoring applications [10], the positioning devices or sensors used to collect information (e.g., location, humidity, and wind speed) also have measurement error.

Due to the inexactness of these data, the quality of applications that run upon them can be affected. A particular flight chosen from the schema integrated using different websites may be associated with a set of possible prices. Out of these possible prices, only one of them is correct. If the system does not know which price is correct, it may choose a wrong price to present to the user. In general, less ambiguous data yield higher service quality. Data ambiguity can be alleviated with the aid of external information. For example, inexact movie ratings can be sanitized by hiring hourly-paid employees who contact users for clarification. In a Location Based Service (LBS), the positions of users may be a set of possible values. The "dirtiness" of this data can be reduced if other "context information" is provided [16] (e.g., if it is known that a user is currently in a meeting, then he should be located in a meeting room). In another example, suppose a district police force has a list of criminal suspects for each unsolved case. The identity of the true offender for each case is ambiguous, since he can be any one of the identified suspects. Through further investigation, suspects can be proven innocent and the identity of the real offender becomes more certain. In this paper, we study the problem of "cleaning" a database of entities (e.g., flights, users, criminal cases) that contain inexact information (e.g., prices, locations, suspects). In particular, we study effective algorithms for selecting good entities to clean, in order to obtain a less-ambiguous or higher-quality database.

We identify four aspects of data cleaning, which commonly arise during the data disambiguation process and affect the effectiveness of cleaning algorithms:

1. Cost. Cleaning an entity often involves a cost. Also, the number of cleaning operations allowed may be limited by a *budget*. In the movie rating database example, employees are paid hourly to make phone calls to the movie-viewers and validate each movie-viewer's ratings. A cleaning cost can also be measured by the time required to clarify the ratings. As another example, there may not be enough police officers to examine all outstanding criminal cases, so the cost of finding the true identity of each offender is also budget-limited.

2. Successfulness. A cleaning activity can fail to remove any ambiguities from an entity. For example, a movie-viewer may be unreachable at a certain time and would require being called back later. To solve a difficult criminal case, an officer may not be able

Proceedings of the VLDB Endowment, Vol. 3, No. 1

to identify the true offender in a single operation.

3. Information availability. If the *successful cleaning probability* of each entity is precisely known, then it is easier to prioritize the cleaning effort (e.g., we can clean an entity with a high probability first). However, in some situations, the probability that an entity can be cleaned may be unknown. For example, the probability that a police officer can remove a suspect from consideration during an investigation may not be available. Instead, we may only know the previous performance of the police officer. This makes it difficult to predict whether a cleaning operation will be successful or not.

4. Incompleteness. A cleaning operation may only be able to remove part of the ambiguities from an entity. For example, after an investigation on a criminal case where a few suspects were identified, a police officer may only be able to eliminate one suspect from further consideration. In a LBS, the use of contextual information may not be able to completely disambiguate a user's location.

Due to the above issues, it is difficult to determine which entities to clean. For example, if an attempt to clean an entity fails, should a cleaning algorithm retry the same entity, believing that it will be successful, or should the algorithm consider another candidate, which might have a higher potential to be cleaned? The decision of either sticking with the old entity or considering a new one is critical, since all cleaning operations are subject to a stringent budget. Our goal is to develop algorithms that can make a "right decision" in selecting entities to clean, which, to the best of our knowledge, has not been considered before.

In this paper, we study two classes of cleaning algorithms. The first algorithm type is *heuristic-based*, and uses some intuitive rules to choose objects to clean. We present three algorithms in this category. For example, in the *greedy algorithm*, the algorithm's success rate on an entity is used to estimate the probability that the said entity can be cleaned. Based on the estimated probabilities and the available budget, one or more entities are chosen. As shown experimentally, these algorithms can often select a good entity to clean.

The heuristic-based algorithms have weaknesses. First, they do not make good use of previous cleaning performance statistics. Second, some parameter values have to be set before execution; however, it is unclear how these parameters should be set to achieve high cleaning effectiveness. Our experiments show that cleaning performance is sensitive to these parameter values. Hence, we develop an algorithm, *Explore-Exploit* (EE), that alleviates these problems. EE considers two decisions: (1) *explore*, or select, a different entity to be cleaned; or (2) *exploit* the entity that has just been cleaned by cleaning it again. In a movie-rating database, an employee may either select a new user to contact (explore), or make another phone call later to the same user who did not answer the phone (exploit). As another example, a police officer may investigate (explore) a new case, or choose to investigate more on his current case (exploit). We study how to appropriately balance the number of exploration and exploitation operations, in order to achieve the highest cleaning effectiveness for the EE algorithm. This algorithm can be executed in linear time and it can support large databases with appropriate data structures. Moreover, EE is applicable to a wide range of databases that manage ambiguous information, e.g., uncertain [8, 20, 21] and probabilistic databases [2, 9, 17]. Our experiments on real and synthetic data illustrate that the EE algorithm is more effective than heuristic-based algorithms in selecting the best set of objects to clean.

The rest of the paper is as follows. Section 2 discusses the related works. In Section 3 we describe the problem definition. In Sections 4 and 5 we present the threshold-based and the EE algorithms respectively. Section 6 presents the experimental results. Section 7 concludes the paper.

2. RELATED WORK

Data cleaning. The problem of disambiguating, or cleaning, a database raises a lot of research interest. In sensor networks, researchers studied how to request, or *probe* data obtained from sensor and data streams [6, 10, 15, 18]. Data probing can be considered as reducing data ambiguity, since the uncertainty of sensor values stored in the system, due to time delay and measurement error, can be reduced by getting new values from the sensors. In [7], the issues of efficiently cleaning a probabilistic database under a stringent cleaning budget are studied. These works all assume that a cleaning operation is always successful. Moreover, in [6, 7], a cleaning action completely removes all ambiguities of an entity (e.g., police always finds the actual criminal after the first investigation). These assumptions may not hold in real applications. In this paper we study the challenging problem of cleaning a large database in practical situations–i.e., without assuming that cleaning is perfect and the probability of successfully cleaning a specific entity is known. We are not aware of any previous work that considers these realistic factors during the data cleaning process.

Duplicate elimination refers to the technique of detecting and deleting similar and inaccurate information in an "inconsistent" database [12]. This can be regarded as a data disambiguation process, where a database becomes cleaner after the deduplication. [1] uses duplicate tuple merging techniques to provide possible answers. In [14], integrity constraints are used for dirty data removal. [4] discusses how to manage different versions of resulting databases due to the duplicate removal process. Our proposed solution can potentially be used in this area, by 1) treating ambiguity information as duplicates and 2) considering practical limitations (e.g., limited budget for eliminating duplicates).

Multi-armed-bandit. The design of the EE algorithm is inspired by a rich vein of work related to the *multi-armed-bandit problem* (bandit-problem for short), a survey of which can be found in Appendix A. In its general form, the bandit-problem describes a gambler at a casino with k slot machines (bandits). Each slot machine, when played by depositing a coin, has an undisclosed probability of giving some monetary reward. Given a limited number of coins, the gambler's goal is to choose a sequence of machines to play, such that the highest expected amount of money can be won [3, 5, 13]. Bandit-problem solutions typically use a simple algorithm framework and make careful decisions on whether to continue playing the same machine, or whether to try another machine. The statistics of each machine (e.g., number of successful plays) facilitate the making of these decisions.

There are plenty of similarities between the bandit-problem and the cleaning problem studied here. Particularly, we can model an ambiguous entity as a bandit. Performing a cleaning operation on the entity is like playing a slot machine, since both actions give "rewards", in terms of the number of ambiguities removed and the amount of money won, respectively. Also, both problems assume that a limited budget is provided and the probability that a specific entity (machine) can be successfully cleaned (played) is not known to the algorithm. Instead, only some general information (e.g., how well cleaning/betting operations did in the past) is known. Hence, we adopt the framework of the bandit-problem solutions (e.g., [3]) in the high-level design of the EE algorithm. However, the bandit-problem solutions cannot be directly adapted to solve our problem. Given a database entity, the maximum amount of ambiguities that can be removed from it is limited. Once all the ambiguities of an entity are removed, that entity is completely clean, and does not need to be considered for further cleaning. This is fundamentally different from bandit-problems, which assume that (1) a slot machine can give unlimited rewards and (2) there are always a fixed

number, k, of slot machines. Due to these differences, the technical details of our solution are substantially different from the bandit-problem solutions.

3. PRELIMINARIES

We first briefly discuss the cleaning solution framework (Section 3.1). Then, we describe the cleaning model in Section 3.2. Finally, we discuss an optimal solution for a special case, in Section 3.3.

3.1 System Framework

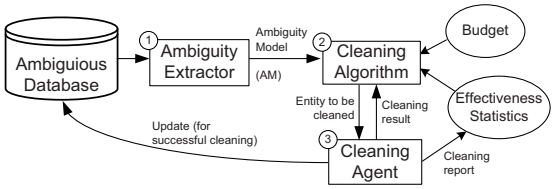


Figure 1: The data cleaning framework.

Figure 1 illustrates the data cleaning framework. Given a database that contains inexact, or erroneous, information, an *ambiguity extractor* derives the corresponding "ambiguity model" (Step 1). This information is then passed to the *cleaning algorithm*, which decides which entity to clean based on a given cleaning budget (Step 2). The decision is then conveyed to the *cleaning agent*, which is responsible for performing the actual cleaning operation (Step 3). If the selected entity can be cleaned, then its ambiguous information is removed from the database. The *effectiveness statistics* of the cleaning agent (e.g., success rate) are also used by the cleaning algorithm. The algorithm and the agent are repeatedly invoked until the cleaning budget is exhausted. A cleaning report, which describes whether a cleaning operation is successful, is used to update the effectiveness statistics.

Notice that in this framework, the components responsible for the cleaning process are independent from the database. Next, we examine the details of this design.

3.2 Ambiguity Model and Cleaning

The **Ambiguity Model** (or *AM* for short) is a succinct summary of ambiguity information collected from a database. It contains a set T of n entities $(T_1, T_2, \ldots, T_n)$. Each entity has a set of values, where only one value is correct. Let r_i be the number of values in T_i (called *false values*) that are actually incorrect. The information in AM can be retrieved easily from databases that store inexact information. For example, the AM uses information from an uncertain database of objects (e.g., NBA players, moving vehicles, and criminal cases), where each object's uncertain attribute (e.g., rebound statistics of an NBA player, location of a vehicle, criminal suspects) is a set of possible values that can be correct [8, 20, 21]. Then, in the corresponding AM, T_i is the i-th uncertain object, and r_i is one less than the number of samples representing an uncertain attribute of T_i. Generally, if T_i has m uncertain attributes (say, $T_i.a_j$), where a_j is the j-th attribute of T_j, then an entity can be created for each (T_i, a_j) pair. As another example, consider a probabilistic database [2,9,17], which contains a set of "x-tuples". Each x-tuple is associated with a group of tuples, which are possible representations of an x-tuple. In this case, T_i is the i-th x-tuple, while r_i is one less than the size of the i-th x-tuple. Since we are only interested in entities with a non-zero number of false values, we assume that $r_i > 0$, for every T_i. Figure 2 illustrates an example AM, which contains five entities with their respective false values.

Entity	No. of false values	successful cleaning probability
T_1	5	0.1
T_2	3	0.4
T_3	6	0.4
T_4	4	0.7
T_5	1	1

Figure 2: Illustrating the ambiguity model.

Cleaning model. Let us now discuss how to model the four aspects of data cleaning discussed in Section 1. Let C be the total number of units (*budget*) available for cleaning entities in T. Each (T_i) has a *successful cleaning probability* (*sc-probability*) p_i, shown in Figure 2. A *cleaning operation* on T_i, $clean(T_i)$, is successful with probability p_i. In many situations, a cleaning operation may fail. For example, a police officer may fail to remove any innocent suspect during an investigation; a GPS device may fail to report its location due to network problems. When $clean(T_i)$ is performed, one cost unit is consumed, and C is reduced by one. Furthermore, if $clean(T_i)$ is successful, one false value is removed from T_i and r_i is decremented by one. Since not all false values are removed in one single execution of $clean(T_i)$, the *incompleteness* factor is also modeled. Although the cleaning model considered here does not model all scenarios, we believe that our solutions can be extended to address other complex models (e.g., a variable number of false values is removed, or new ambiguity information is introduced to the entity during the disambiguation process). We plan to study these issues in our future work.

To model the *information availability* aspect of data cleaning, we assume that the cleaning algorithm does not know how likely the cleaning agent can remove a false value from each entity (i.e. the cleaning algorithm does not know p_i). Instead, it is provided with some statistics about the cleaning agent's performance. A kind of statistic used by the EE algorithm is the *sc-pdf*, the probability distribution function (pdf) of sc-probabilities. The sc-pdf contains information about how the cleaning agent performs, which can be collected from the agent's previous cleaning performance. For example, Figure 3 shows the sc-pdf for Figure 2. In EE, f can be a continuous pdf (e.g., Gaussian or uniform distribution), or an arbitrary histogram represented as an array of values.

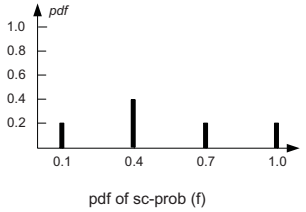


Figure 3: successful probability distribution (sc-pdf).

Effectiveness of cleaning algorithms. The algorithms that we present next return a sequence of cleaning operations. To measure the effectiveness of these algorithms, let A be a cleaning algorithm and R be the total number of false values removed by A. We define the *cleaning effectiveness* of A as:

$$\xi(A) = \frac{E[R]}{C} \quad (1)$$

where $E[R]$ is the expected number of false values removed by A. Intuitively, the higher the value of $\xi(A)$, the more beneficial a cleaning operation is. We will use Equation 1 to measure and compare the effectiveness of different algorithms. Table 1 summarizes the notations used in the paper.

3.3 Solution for Known sc-probability

Notation	Meanings
C	total cleaning budget
T	set of n entities $(T_1, \ldots, T_n)$
r_i	total no. of false values of T_i
$clean(T_i)$	cleaning operation on T_i
p_i	sc-probability of T_i
f	sc-pdf
R	no. of false values removed by cleaning algorithm
$\xi(A)$	cleaning effectiveness of algorithm A

Table 1: Symbols used in this paper.

We conclude this section with the discussion of a simple algorithm, called `known-sc`, that handles the situation when the sc-probability values are known. This algorithm is used as a reference that other algorithms, with unknown sc-probability information, are compared with, experimentally. In `known-sc`, we first sort the entities in descending order of sc-probabilities. Then, for each entity in this order, we remove all their false values first, before visiting the next entity, until the cleaning budget is exhausted.

Without stating the detailed proof, we claim that this algorithm is optimal in effectiveness. This is because an entity with a higher sc-probability should always be cleaned ahead the one with a lower sc-probability, in order to maximize the expected number of false values removed. Hence, the knowledge of sc-probabilities facilitate effective cleaning. However, sc-probabilities are not always known. To address this problem, we present the heuristic-based algorithms (Section 4) and the `EE` algorithm (Section 5).

4. HEURISTIC-BASED ALGORITHMS

Let us now describe three heuristic-based algorithms, `Random`, `Greedy`, and ϵ`-Greedy`. These algorithms, as described in (Sections 4.1, 4.2, and 4.3), use simple observations to select an entity to clean. They also provide the foundations for understanding the more sophisticated `EE` algorithm.

4.1 The Random Algorithm

Our first cleaning algorithm is called `Random` (Appendix B). This solution randomly chooses an entity T_j from the AM (Line 2). After *clean*(T_j) is executed, the budget, C, is decremented (Line 5). The process repeats until either C becomes zero or no more entities have false values.

The design of `Random` is based on the *fairness* principle: the algorithm gives every entity an equal chance to be cleaned. This can be useful in situations when it is not clear which entity will offer a better chance to have its false values removed. The algorithm is also simple to implement. However, `Random` does not consider the fact that some entities' false values are easier to be removed; for these entities, it is natural to invest more cleaning effort, as illustrated by the next algorithm.

4.2 The Greedy Algorithm

In the `known-sc` algorithm (Section 3.3), the sc-probability of an entity (p_i) is useful for removing false values. The main idea of the `Greedy` algorithm (Appendix B) is to estimate sc-probabilities from previous cleaning results. Entities are then selected in descending order of these estimated values–i.e., the entity selection is *greedy* on the estimated sc-probabilities. To illustrate, let $\hat{p}_i \approx p_i$, and let t_i and s_i be, respectively, the number of cleaning operations on T_i and the number of successful cleaning operations on T_i. Then, $\hat{p}_i = s_i/t_i$. If more than one entity has the same $\hat{p}_i$, we randomly pick one among them. The entity with the largest $\hat{p}_i$, which still has false values, is selected. This process is repeated until no more budget is available, or all entities have been cleaned.

While `Greedy` provides a simple way of approximating p_i, the problem is that the $\hat{p}_i$'s so estimated may not be statistically correct. For instance, suppose that a first cleaning attempt on entity T_1, with $p_1 = 0.01$, is successful. Then, $\hat{p}_1 = s_1 = t_1 = 1$. `Greedy` will then assume T_1 is a good candidate to clean next time, even though p_1 is actually small. Under a stringent budget, it may be better to move on to other entities that have higher sc-probabilities. The next algorithm provides a "quick fix" to this problem.

Algorithm	Time cost	Space cost
`Random`	$O(M \cdot C)$	$O(n)$
`Greedy`	$O((n+C)\log n)$	$O(n)$
ϵ`-Greedy`	$O(Cn)$	$O(n)$
`EE`	Initialization: $O(\lvert T\rvert + \frac{\hat{t}^3}{\delta^2})$ Explore+Exploit: $O(C)$	$O(n)$

Table 2: Time and space costs of cleaning algorithms.

4.3 The ϵ-Greedy Algorithm

Conceptually, the ϵ`-Greedy` algorithm (Appendix B) combines the essence of `Random` and `Greedy`, and alleviates the problem of `Greedy` by allowing the entity selection process to find new candidates. Here, we introduce a parameter ϵ, which is a real value in $[0, 1]$. A random variable, $choice \in [0, 1]$, is first generated (Line 3). If $choice < \epsilon$, a random entity is selected (Lines 4-5). Otherwise, the entity with the highest estimated sc-probability ($\hat{p}_i$) is chosen (Line 7). The process is repeated until the budget is exhausted, or no ambiguous entity remains.

This algorithm avoids the problem of wrongly estimating sc-probabilities, since there is some chance (controlled by ϵ) that an entity is randomly chosen, so that some entity with a high cleaning probability, which may be missed by `Greedy`, can still be chosen. The main disadvantage is that ϵ is a user-defined parameter and setting ϵ can be crucial to the cleaning effectiveness. Our experimental results show that cleaning effectiveness is indeed sensitive to ϵ. On the contrary, the `EE` algorithm, discussed in the next section, does not require parameter tuning.

Table 2 summarizes the time and the space complexities of the heuristic-based algorithms. Their derivation details can be found in Appendix B. With appropriate data structures (e.g., use of arrays and priority queues), these algorithms are efficient and require only a small amount of memory.

5. THE EXPLORE-EXPLOIT ALGORITHM

We now present the *Explore-Exploit* (`EE`) algorithm. We first discuss a general framework in Section 5.1. Then, Section 5.2 proposes the `EE` algorithm under this framework. In Section 5.3 we explain how to adjust parameters for `EE` to achieve optimal effectiveness. We discuss a practical way of adjusting these parameters in Section 5.4. In Section 5.5 we discuss a way to enhance `EE`.

5.1 Solution Framework

Let us now present a solution framework, which generalizes the `EE` algorithm.We establish some theoretical results about this framework, which will be used to develop `EE`. This study also facilitates future development of other algorithms with a similar structure.

Algorithm 1 `Framework`(AM T, Budget C)

```
1: while C > 0 do
2:     randomly choose an entity T_i where r_i > 0
3:     explore(T_i, C)  // Quit if C = 0
4:     if C > 0 and condition for running exploit is satisfied
           then
5:         exploit(T_i, C)  // Quit if C = 0
```

Notation	Description
γ_i	no. false tuples removed from T_i in 1 round of Framework
χ_i	cleaning cost on T_i in 1 round of Framework
Γ	$\frac{E[\gamma_i]}{E[\chi_i]}$; approximation of ξ(Framework)
$t, \hat{t}$	Exploration frequency and its maximum value
$q, \hat{q}$	Exploitation threshold and its approximation
δ	Precision used in numerical integration
(t_{max}, q_{max})	(t, q) that maximizes ξ(EE)

Table 3: Notations used in the EE algorithm.

As we can see from Framework (Algorithm 1), during each iteration an entity T_i, which still has false values, is chosen randomly (Line 2). Then, an *explore* procedure is applied on T_i, which performs some number of cleaning operations on T_i, in order to get information about how likely T_i's false values can be removed (Line 3). If there is still some budget left, and the algorithm considers it worthy to invest more effort on T_i, an *exploit* procedure is performed, where more cleaning operations are performed on T_i (Lines 4-5). The whole process is repeated until the budget is exhausted. Notice that *explore* and *exploit* may be stopped while they are being run (if there is not enough budget to complete the two operations). We say that T_i is *fully visited*, if *explore* and *exploit* are completely executed on T_i during an iteration without being interrupted due to the lack of budget.

The exact details of *explore* and *exploit* depend on the specific algorithm (e.g., EE). We also remark that a similar scheme is also used in solving the *multi-armed-bandit* problem [3]. Table 3 displays the symbols used in this section.

Cleaning effectiveness. We now present some theoretical results about Framework, which enables its cleaning effectiveness (Equation 1) to be estimated. Suppose after the execution of Algorithm 1, exactly k entities have been fully visited. Let R_k be the total number of false values removed from these k entities, and C_k be the total cleaning effort invested on them. Then, we have the following lemma.

LEMMA 1. $\lim_{C\to\infty} \xi(\texttt{Framework}) = \lim_{k\to\infty} E[\frac{R_k}{C_k}]$

Lemma 1 tells us that the cleaning effectiveness of Framework (i.e., ξ(Framework)) can be expressed in terms of R_k and C_k. The proof is based on the fact that the $(k+1)$-th entity cannot be fully visited before the budget is exhausted, so the number of ambiguities that can be removed from T_{k+1} is small. The details of the proof can be found in Appendix C. We then present the following result:

LEMMA 2. *Let γ_i be the total number of false values removed from T_i after it has been fully visited. Also, let χ_i be the sum of cleaning costs required for removing these false values. Then,*

$$\lim_{C\to\infty} \xi(\texttt{Framework}) = \frac{E[\gamma_i]}{E[\chi_i]} \quad (2)$$

Essentially, Lemma 2 states that the effectiveness of Framework is simply an expression of the expected values of γ_i and χ_i (i.e., $E[\gamma_i]$ and $E[\chi_i]$). The proof is based on the observation that γ_i and χ_i are independent variables, and that R_k and C_k can be expressed as functions of γ_i's and χ_i's. By using the law of large numbers and Lemma 1, the lemma can be proved. The details are described in Appendix C.

Now, Let Γ(Framework) be the constant $\frac{E[\gamma_i]}{E[\chi_i]}$, the effectiveness of the Framework. For the sake of convenience, we use Γ to represent Γ(Framework). The following equation can then be used to estimate ξ(Framework):

$$\xi(\texttt{Framework}) \approx \Gamma \quad (3)$$

We will also use Equation 3 to decide the optimal parameter values of EE, as discussed next.

5.2 The EE Algorithm

The EE algorithm is a specialization of Framework. The main phases of EE, as detailed in Algorithm 2, include:

1. Initialization. The values of two parameters, namely the *exploration frequency* (t) and the *exploitation threshold* (q), are computed. The exploration frequency is the number of times an entity is cleaned during exploration, while the exploitation threshold controls whether exploitation on an entity is executed. The values of t and q are set in a way such that the cleaning effectiveness of EE is optimized, as detailed in Section 5.3.

2. Exploration. This corresponds to *explore*(T_i, C) in Framework. Here, a randomly chosen entity T_j is cleaned t times. We also collect some statistics: m (the number of false values removed for T_i so far) and d (the number of times that *clean*(T_j) is executed) (Lines 8-12). After cleaning is completed in this phase, we compute the approximate sc-probability $\hat{p}_i$ of T_i, as m/t (Line 15).

3. Exploitation. This corresponds to *exploit*(T_i, C) in Framework. First, $\hat{p}_i$ is tested if it is larger than q, the exploitation threshold (Line 17). If this is false, T_i will not be considered anymore (Line 23). Otherwise, *clean*(T_i) is repeatedly executed, until either no budget is available, or all the false values have been removed (i.e., $r_i = 0$) (Lines 18-21).

Algorithm 2 EE(AM T, Budget C, sc-pdf f)

```
1: [1. Initialization]
2: Find (t, q) such that Γ(T, f, t, q) is maximized
3: while C > 0 do
4:     randomly choose an entity T_i, where r_i > 0
5:     [2. Exploration Phase (Lines 6-15)]
6:     m ← 0, d ← 0
7:     while C > 0 ∧ d < t ∧ r_i > 0 do
8:         if clean(T_i) is successful then
9:             r_i ← r_i − 1
10:            m ← m + 1
11:        C ← C − 1
12:        d ← d + 1
13:    if C = 0 then
14:        Quit
15:    p̂_i ← m/t
16:    [3. Exploitation Phase (Lines 17-23)]
17:    if p̂_i ≥ q then
18:        while C > 0 ∧ r_i > 0 do
19:            if clean(T_i) is successful then
20:                r_i ← r_i − 1
21:            C ← C − 1
22:    else
23:        Do not consider T_i anymore
```

Next, we examine how to optimally set the values of the parameters (t, q) of EE.

5.3 Setting Parameters of EE

The parameter values in Line 2 of EE (i.e., t and q) are crucial to the cleaning effectiveness of EE (i.e., ξ(EE)). Specifically, t, the exploration frequency, controls the number of cleaning actions during exploration. The exploitation threshold, q, decides whether exploitation for T_i can take place, so that further cleaning actions can occur. Hence, it is important to find good values of t and q so that ξ(EE) can be maximized.

Recall from Lemma 2 that $\xi(\texttt{EE}) = \Gamma$, where $\Gamma = \frac{E[\gamma_i]}{E[\chi_i]}$. Hence, the values of t and q should be set in a way that yields a maximum value of $\frac{E[\gamma_i]}{E[\chi_i]}$. To achieve this, suppose $E[r_i]$ is the expected number of false values per entity T_i. Notice that $E[r_i]$ can be found by averaging the number of false values of all entities in T. Then, the following lemma presents the expressions of $E[\gamma_i]$ and $E[\chi_i]$, which allow optimal values of t and q to be found.

LEMMA 3. *$E[\gamma_i]$ and $E[\chi_i]$ can be expressed as:*

$$E[\gamma_i] = \int_0^1 (P_{ne}(p)E_t(p) + (1 - P_{ne}(p))E[r_i])f(p)dp \quad (4)$$

$$E[\chi_i] = \int_0^1 (P_{ne}(p) \cdot t + (1 - P_{ne}(p)) \cdot \frac{E[r_i]}{p})f(p)dp \quad (5)$$

where $P_{ne}(p)$, the chance that an entity with sc-probability p is explored but not exploited, is:

$$P_{ne}(p) = \begin{cases} \sum_{m=0}^{\lceil tq \rceil - 1} C_m^t p^m (1-p)^{t-m} & E[r_i] > \lceil tq \rceil - 1 \\ 1 & otherwise \end{cases} \quad (6)$$

and $E_t(p)$, the expected number of false values removed from an entity with sc-probability p after exploration and before exploitation, is:

$$E_t(p) = min(tp, E[r_i]) \quad (7)$$

PROOF. (**Sketch**) For **Equation 4**, let the number of false values removed from some entity T_i with sc-probability p be $V(p)$. Then, $E[\gamma_i]$, the expected amount of false values removed, is an integration of $V(p) \cdot f(p)$ over $[0, 1]$. To obtain $V(p)$, we consider two cases of T_i:

- **Case 1:** With probability $P_{ne}(p)$, T_i is only explored, but not exploited. Then T_i has $E_t(p)$ false values removed.
- **Case 2:** With probability $(1 - P_{ne}(p))$, T_i is both explored and exploited. Since this entity has an expected number $E[r_i]$ of false values, using the EE algorithm, $E[r_i]$ false values will be removed.

Then, $V(p)$ can be expressed in terms of $E[r_i]$, $P_{ne}(p)$, and $E_t(p)$. The remaining tasks are to find $E_t(p)$ and $P_{ne}(p)$. We obtain $E_t(p)$ (Equation 7) by observing that the expected number of false values that can be removed after exploration is bounded by the minimum of $t \cdot p$ and $E[r_i]$. For $P_{ne}(p)$, notice that if an entity, which has been explored, is not exploited, the condition that $\hat{p}_i \geq q$ in Line 17 should fail. Since $\hat{p}_i = m/t$ (Line 15), we can obtain the maximum value of m that cannot satisfy the condition in Line 17. We then express the probability that m cleaning operations fail out of t trials, as a binomial distribution, and use it to express $P_{ne}(p)$, which results in Equation 6.

We derive **Equation 5** in a similar manner. Let the amount of cleaning effort spent on an entity T_i with sc-probability p be $S(p)$. Then, $E[\chi_i]$, the expected cleaning cost on each entity is an integration of $S(p) \cdot f(p)$ over $[0, 1]$. To obtain $S(p)$, we consider two cases for T_i:

- **Case 1:** With probability $P_{ne}(p)$, T_i is only explored, but not exploited. Then T_i needs a cost of t to be cleaned.
- **Case 2:** With probability $(1 - P_{ne}(p))$, T_i is both explored and exploited. Notice that this entity has an expected number $E[r_i]$ of false values. Also, on average, a number $\frac{1}{p}$ of cleaning operations are needed to remove one false value from T_i. Using EE, a cost of $\frac{E[r_i]}{p}$ is required to remove $E[r_i]$ false values from T_i.

Hence, $S(p)$ can be expressed in terms of $E[r_i]$ and $P_{ne}(p)$. By using the formula of $P_{ne}(p)$ (Equation 6), Equation 5 is obtained. □

The detailed proof of Lemma 3 can be found in Appendix C.

By using Lemma 3, if t and q can be found such that $\Gamma = \frac{E[\gamma_i]}{E[\chi_i]}$ is optimal, then the highest effectiveness of EE is obtained. However, it is difficult to find a closed-form formula of t and q. We next discuss a practical method of finding t and q.

5.4 Practical derivation of parameters

We find the best pair of (t, q) by performing the following steps:
Step 1. Derive a statistical upper bound of t, denoted as $\hat{t}$, which is equal to:

$$\hat{t} = \frac{1}{E[p_i]} \cdot E[r_i] \quad (8)$$

Note that the expected sc-probability $E[p_i]$ of an entity T_i can be computed by $E[p_i] = \int_0^1 pf(p)dp$. Hence, $\frac{1}{E[p_i]}$ is the average number of cleaning operations required to remove one false value from T_i. Equation 8 depicts the expected number of cleaning operations used to remove the average number of false values of an entity.
Step 2. Let $\delta \in [0, 1]$ be a real-valued parameter, which is used to approximate q. For each integer in $[1, \hat{t}]$, we iterate over a fixed number $\lceil 1 + \frac{1}{\delta} \rceil$ of real values $\hat{q} \in [0, \delta, 2\delta, \ldots, (i-1)\delta, 1]$, and find the corresponding Γ, with Equation 3.
Step 3. Return the pair (t_{max}, q_{max}) that yields the highest value of Γ.

The above procedure limits the number of (t, q) pairs that need to be examined. Our experiments show that the value of δ do not need to be very small, and this method can often find a close-to-optimal value of Γ (and hence a high cleaning effectiveness for EE).

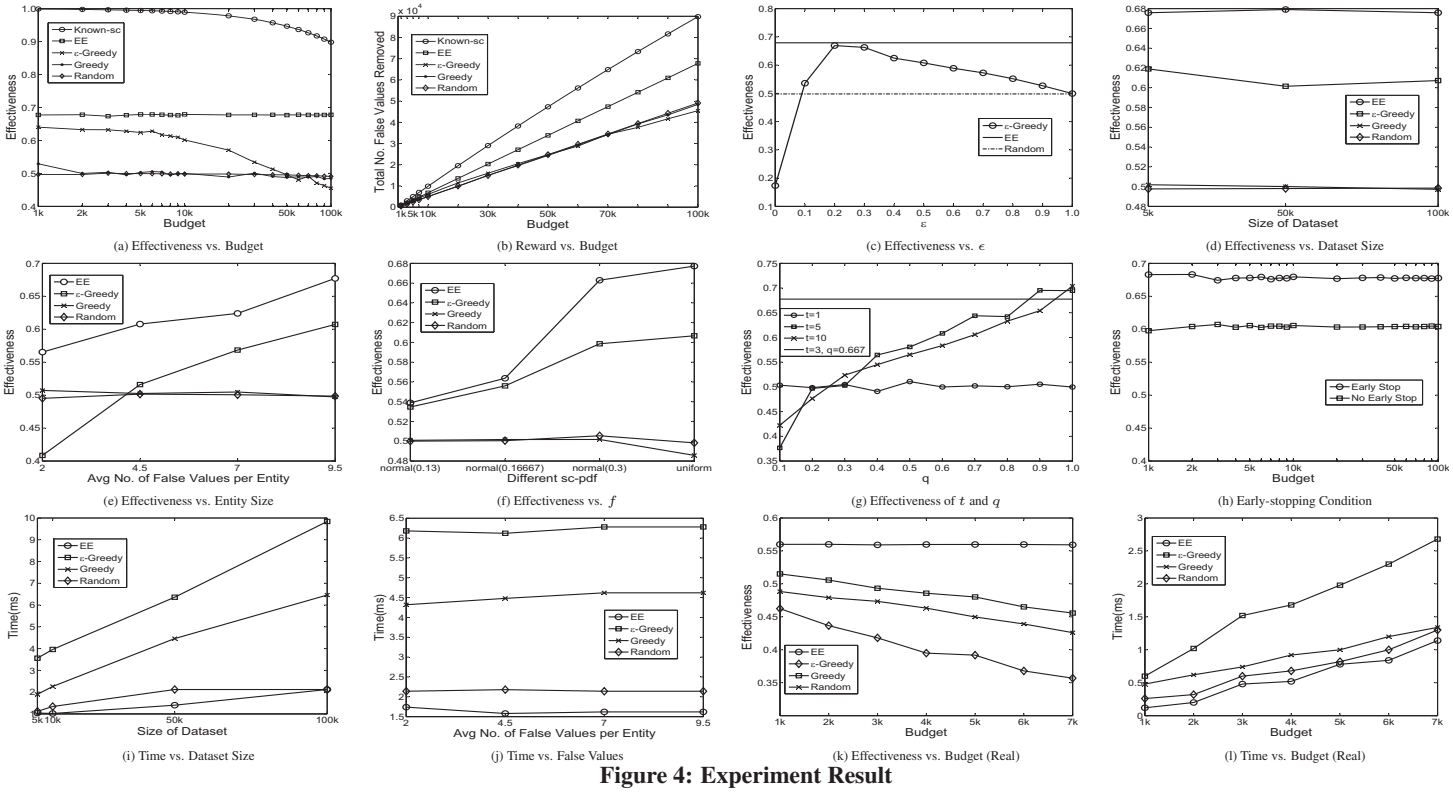
5.5 Stopping the Exploration Early

We conclude this section with the discussion of how to further improve the effectiveness of EE. Recall that the exploitation phase stops when the number of cleaning operations exceeds t (Line 7). This is not always necessary. Observe that the number of cleaning operations that fail to remove false values during exploration is $d - m$. In order for this entity to be exploited, the condition in Line 17 (i.e., $\frac{m}{t} \geq q$) must be satisfied. This implies:

$$\frac{d - m}{t} < 1 - q \quad (9)$$

Once Equation 9 is violated, we can conclude that since too many cleaning operations fail for the entity being considered, the estimated sc-probability must be too low for exploitation to go on. Hence, there is no need to further explore this entity and it can be discarded by EE right away. We call Equation 9 the *early-stopping condition*. Notice that this condition can be applied even if the practical method discussed in Section 5.4 is used, by replacing (t, q) by (t_{max}, q_{max}) in Equation 9. The effect of using the early-stop condition in EE is examined in our experiments.

Complexity. Table 2 shows the time and space complexities of the EE algorithm. While the time complexity of parameter initialization (Phase 1) seems to be large, in practice, $\hat{t}$ is small (only 20 in our experiments). Moreover, if the metadata about T is available (e.g., sc-pdf and $E[r_i]$), initialization can be done offline. The detailed complexity analysis can be found in Appendix B. Finally, notice that the ambiguity model is a compact representation of the database, which can be assumed to be stored in the main memory. If the whole database is stored in disk, then during each explore-exploit iteration of EE, only the information of one entity needs to



(a) Effectiveness vs. Budget (b) Reward vs. Budget (c) Effectiveness vs. ϵ (d) Effectiveness vs. Dataset Size

(e) Effectiveness vs. Entity Size (f) Effectiveness vs. f (g) Effectiveness of t and q (h) Early-stopping Condition

(i) Time vs. Dataset Size (j) Time vs. False Values (k) Effectiveness vs. Budget (Real) (l) Time vs. Budget (Real)

Figure 4: Experiment Result

be retrieved from the disk. Hence, the number of I/Os required by EE is O(n).

6. EXPERIMENTS

To validate the effectiveness and performance of our solutions, we used a synthetic and a real dataset as ambiguity models (AMs) in our experiments. For the synthetic dataset, the number of entities, T, is $50k$, and the number of false values per entity is uniformly distributed in $[0, 19]$, i.e., each entity has 9.5 false values on average. We also retrieve the AM from a probabilistic database, which stores uncertainty about movie-viewer ratings [23]. In this AM, an entity is a (customer,movie) pair, whereas the false values are the customer's possible ratings on that movie. This AM has 4,999 entities, and each entity has one false value on average.

By default, for both datasets, the sc-pdf is a uniform distribution. The value of ϵ for the $\epsilon-$`Greedy` is 0.5, while the value of δ that we used to find t_{max} and q_{max} in `EE` is 0.001. For the synthetic dataset, the default budget is $10K$, and for the real dataset, the budget is $5K$. Unless stated otherwise, the results for the `EE` algorithm that uses the early-stopping condition (Section 5.5) are presented.

All our algorithms were implemented in C++ and run on a 2.66GHz Intel Duo PC with 2GB of memory and Windows 7. Next, we present the results for synthetic and real datasets in Sections 6.1 and 6.2.

6.1 Results on Synthetic Data

1. Cleaning effectiveness. Figure 4(a) shows the cleaning effectiveness, ξ (Equation 1), of different cleaning algorithms. The algorithm `known-sc`, where the exact sc-probability of each entity is known, is also shown for comparison. As we can see, the effectiveness of `known-sc` is the highest. This is not surprising. As discussed in Section 3.3, `known-sc` can make use of sc-probabilities to select the best entities to clean. However, when exact sc-probabilities are not known, we can see that the `EE` algorithm is the most effective over a wide range of budgets. At a budget of $10k$, for instance, `EE` is, respectively, 12.9%, 35.8%, and 36.4% better than ϵ-`Greedy`, `Greedy`, and `Random`. The effectiveness of `EE` is also stable over the change of budget values. To understand why, Figure 4(b) shows the corresponding number of false values removed. We can see that the number of ambiguities removed increases almost linearly with the budget, for `EE`. This explains the stability of `EE`. From now on, we focus on the cleaning algorithms where exact sc-probabilities are not known.

We observe that heuristic-based algorithms such as `Random` and `Greedy` are the least effective. This reflects that they cannot select the best entities. The effectiveness of $\epsilon-$`Greedy`, on the other hand, vary significantly: it performs close to `EE` with a small budget, but degrades as the budget increases. For $\epsilon-$`Greedy` to work, the value of ϵ is set by the user first. In Figure 4(a), the default value of ϵ is always used regardless of the budget available. Figure 4(c) shows that the effectiveness of ϵ-Greedy is sensitive to the value of ϵ. Hence, it is difficult to find a fixed value of ϵ that works well for every budget value. `EE` does not suffer from this problem; in fact, `EE` is stable over a wide range of budgets.

2. Effect of dataset sizes. Figure 4(d) examines the effectiveness of cleaning algorithms under different dataset sizes (number of entities). As we can see, `EE` is consistently the best over a wide range of dataset sizes. At a size of $100k$, `EE` is, respectively, 11.3%, 35.9%, and 35.6% better than ϵ-`Greedy`, `Greedy`, and `Random`.

3. Effect of false values. Next, Figure 4(e) shows how the change of the average number of false values per entity affects cleaning effectiveness. `EE` still performs the best. Moreover, its effectiveness increases with the number of false values. With the increase of the number of ambiguities, `EE` has more opportunity to decide whether an entity should be cleaned during exploration before all the false values are removed. The valuable information collected from exploration can further be used to decide whether the remaining false values to be handled during exploration. Thus,

EE has higher effectiveness with more false values.

4. Effect of sc-pdf. We then examine in Figure 4(f) how the variation of sc-pdf can affect effectiveness. Here, we use the notation $normal(\sigma)$ to say that a sc-pdf has a normal distribution with mean 0.5 and variance σ. For comparison, we also display the result for uniform sc-pdf. EE is more effective than heuristic-based algorithms. Its effectiveness also increases with σ. Conceptually, a larger σ indicates that it is harder to guess the actual sc-probability value, and represents a more uncertain cleaning scenario. As we can see, EE works particularly well under this setting. For example, at $\sigma = 0.3$, EE is, respectively, 10.7%, 32.1%, and 33.1% better than ϵ-Greedy, Greedy, and Random.

5. Other experiments on EE. Figure 4(g) shows the effect of different combinations of the parameter values (i.e., t and q) used by EE. Generally, for the same value of t, the effectiveness increases with q. In the same figure, we also indicate the effectiveness under $t_{max} = 3$ and $q_{max} = 0.667$, found by the approximate method described in Section 5.4. We can see that the effectiveness found by our method is close to the optimal value. In Figure 4(h), we observe that the use of the early-stopping condition in the exploration phase (Section 5.5) enhances the effectiveness of EE by an average of 22.08%.

6. Performance analysis. Next, we examine the time required by the algorithms to decide which entities to clean. For EE, the time for initializing the parameters t and q is 5.99 seconds on average. We assume this to be done offline, and in the figures we present the online time required by EE (for exploration and exploitation). We can see from Figure 4(i) that all algorithms can be finished below 10ms. They also work well for large datasets. We see that ϵ-Greedy performs the worst because it has to maintain two data structures (array and priority queue). Figure 4(j) shows that EE is very efficient over a wide range of the number of false values.

6.2 Results on Real Data

Since most experiments on the real dataset show a similar trend as the synthetic data, we only present the most representative ones.

7. Effectiveness vs. budget. We examine the impact of varying the cleaning budget on effectiveness in Figure 4(k). As we can see, the effectiveness of the heuristic-based algorithms drops with an increase in the cleaning budget. This means they do not make good use of the extra budget available. On the other hand, EE is stable over different budget values, showing that EE can adapt itself to different budget values.

8. Performance vs. budget. Finally, Figure 4(l) shows the running time required by different algorithms. Due to the presence of a larger budget, more time is needed for all algorithms. However, they scale well with the budget, and take less than 10ms to finish. Moreover, we observe that EE has a low running time compared with other algorithms.

7. CONCLUSIONS

The management of ambiguous, inexact, and uncertain databases has become important in new and emerging applications. We study algorithms for disambiguating these data in realistic scenarios: cleaning is budget-limited, exact sc-probability is not known, and cleaning is partially completed. We propose algorithms based on simple heuristics, as well as the more sophisticated EE algorithm. An advantage of EE is that its parameters can be determined automatically, according to the information provided by the ambiguity model. Our experiments show that the effectiveness of EE is consistently the highest among our proposed algorithms under different settings. Moreover, all these algorithms are simple, efficient, and scalable to large databases.

We plan to extend our solutions to support other environments where the data disambiguation process is not perfect. For instance, it is interesting to address the case when an entity has some ambiguities again after it has been cleaned (e.g., a sensor value becomes outdated after it has been probed from sensor sources). We will also study scenarios where the number of false positives that can be removed, as well as the cleaning cost, vary across different entities. As mentioned in Section 5, *EE* is only a specific algorithm under a general *multi-armed-bandit* algorithm framework. We plan to examine other variants of *EE* that also fit under this framework.

8. ACKNOWLEDGEMENTS

Reynold Cheng was supported by the Research Grants Council of Hong Kong (RGC Projects HKU 513508 and 711309E). Eric Lo and Ming-Hay Luk were supported by the RGC Projects PolyU 525009E. We would like to thank Raymond Wong (HKUST) for offering his advice in the early phase of the work. We also thank the reviewers for their insightful comments.

9. REFERENCES

[1] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[2] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. 4(5), 1992.

[3] D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, 1985.

[4] G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. Modeling and querying possible repairs in duplicate detection. *VLDB*, 2009.

[5] D. Chakrabarti, R. Kumar, F. Radlinski, and E. Upfal. Mortal Multi-Armed Bandits. In *NIPS*, 2008.

[6] J. Chen and R. Cheng. Quality-aware probing of uncertain data with resource constraints. In *SSDBM*, 2008.

[7] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. *VLDB*, 2008.

[8] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[10] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[11] H.-H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *VLDB*, 2002.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.

[13] J. C. Gittins and D. M. Jones. A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics*, 1974.

[14] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDE*, 2006.

[15] Z. Liu, K. C. Sia, and J. Cho. Cost-efficient processing of min/max queries over distributed sensors with uncertainty. In *ACM SAC*, 2005.

[16] M. Grossmann et al. Efficiently managing context information for large-scale scenarios. In *PerCom*, 2005.

[17] O. Mar, A. Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *VLDB*, 2006.

[18] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.

[19] P. Sistla et al. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*. 1998.

[20] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

[21] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

[22] Stanford University Infolab. Crime dataset. http://infolab.stanford.edu/trio/code/crime_data.triql.

[23] Stanford University Infolab. Moving rating database. http://infolab.stanford.edu/trio/code/movie_data.triql.

APPENDIX

A. A SURVEY OF MULTI-ARMED BANDIT PROBLEMS

The multi-armed bandit problem [2] is a kind of sequential task scheduling problem. In multi-armed bandit problems, we are presented with a gambler, who needs to maximize the sum of rewards from a slot machine, which is also called a one-armed bandit. The slot machine in multi-armed bandit problems has multiple levers, independent of each other, that the gambler may choose to pull. Each lever (arm) of the bandit has an associated probability distribution (unknown to the gambler) of giving a reward. Furthermore, the gambler has a fixed budget limiting the number of times that he can pull any arm of the bandit. The goal of a multi-armed bandit problem is to develop a sequence of pulls that the gambler should follow, in order to maximize the rewards obtained by playing the bandit. We now present a brief survey of the works in this area. Please refer to Section D for references cited in this section.

The classic multi-armed bandit problem, as stated above, is shown to have an optimal policy [6]. The optimal policy for the classic multi-armed bandit problem is an "index-based policy". For each arm of the bandit, the value known as the Gittins index is calculated from a function of the arm's current state and the expected reward that the arm will yield. That is, the Gittins index measures the maximum obtainable reward for a given arm. As such, the optimal policy is for the gambler to select the lever that currently has the highest Gittins index value.

The classic multi-armed bandit problem assumes that the bandit has a constant set of k arms. In the arm-acquiring bandits variation [10], new arms are added to the bandit over time. The Gittins index based policy was shown to still be optimal in this variation [10].

Further extensions to the classic multi-armed bandit problem have received much interest over the years. One such extension was the proposal of the restless bandit problem [11]. In the classical multi-armed bandit model, the lever that is pulled will experience a change in its state. The restless bandit problem extends the classical multi-armed bandit problem to allow all arms of the bandit to change their respective states even when the arm was not selected to be pulled. Papadimitriou and Tsitsiklis [8] showed that finding an optimal scheduling policy to the restless bandit problem is PSPACE-hard.

Another variation on the multi-armed bandit problem is one where the bandit has dependent arms [7]. In this variation of the classic problem, certain arms of the bandit generate similar rewards. The dependent armed bandit problem inherently forms groups of arms with similar behavior. Pandey et al. [7] showed that an optimal policy exists to solve the dependent armed bandit problem.

Chakrabarti et al. proposed a variation modeling mortal bandits [4]. Mortal bandits are bandits with arms that have a specific lifetime after which they are unavailable to the gambler. When an arm becomes unavailable, a new arm will be made available in place of the one that has just expired. As such, there is a constant set of k arms available to the gambler, some of which the gambler may not have pulled before – these are the arms that replace expired arms. The mortal bandit problem is shown to have an optimal policy [4].

Other variations to the classic multi-armed bandit problem include Brownian restless bandits [9], sleeping bandits [3, 5], and bandits with metric switching costs [1]. Brownian restless bandits are similar to restless bandits. However, Brownian bandits evolve at a rate following Brownian motion. As such, Brownian bandits evolve at a rate that is less wildly random than restless bandits [11]. Sleeping bandits are similar to mortal bandits. Contrasting with mortal bandits, where arms expire, sleeping bandits have arms that will periodically be unavailable, only to be available again at a later time. Finally, bandits with metric switching costs [1] impose a penalty when the gambler chooses to pull a different arm than the one that was last pulled.

B. COMPLEXITY ANALYSIS AND PSEUDOCODES

We now describe the time and space complexities of the cleaning algorithms.

- `Random` (Section 4, Algorithm 3): the pseudocode of `Random` is shown in Algorithm 3. We first put all entity IDs randomly into a sized-n array, in $O(n)$ time. During each iteration, when Line 2 is executed, a random value m is retrieved from $[0, n]$. The m-th entity ID is then retrieved from the array and cleaned, in $O(1)$ time. If this entity has no false values, another random number is generated, for a fixed number M (say, $M = 1000$) times, or until an entity with non-zero number of false values is found. If after M times, no entities with non-zero number of false values are found, the algorithm quits. During each iteration, and the budget C will be reduced by one. Thus, the time complexity is $O(M \cdot C)$, and the space required to store the array is $O(n)$.
- `Greedy` (Section 4, Algorithm 4): the pseudocode of `Greedy` is shown in Algorithm 4. We use a priority queue structure (which is a heap) to maintain the approximate sc-probability values (i.e., $\hat{p}_i$). First, we insert n pairs of (T_i, p_i) into the structure, which can be done in $O(n \log n)$ time. When Line 3 is executed, the entity with the highest sc-probability (say, T_m) is removed the structure, in $O(\log n)$ time. After cleaning, sc-probability is updated, if T_m still has some false values, it is reinserted to the structure, in $O(\log n)$ time. Hence, the time complexity is $O((n + C) \log n)$, and the space complexity is $O(n)$.
- $\epsilon-$`Greedy` (Section 4, Algorithm 5): the pseudocode of $\epsilon-$`Greedy` is shown in Algorithm 5. Since this algorithm is a "hybrid" form of `Random` and `Greedy`, we use the data structures of `Random` and `Greedy` to support Lines 5 and 7 with slight modifications. If an entity has no more false values, then the entity is marked "invalid" in the priority queue (by a linear scan in $O(n)$ time). Later when this invalid entity is popped from the priority queue, it is ignored. At each iteration, a random selection (Line 5) costs $O(1)$, while a greedy selection (Line 7) involves $O(2 \log n)$. Since there are at most one entity with false values that can be removed in one iteration, at most $O(n)$ time is needed to remove an entity from the priority queue. Hence, the cost of each iteration is $O(\max(1, 2 \log n) + n) = O(n)$. Since there are C iterations, the total time cost is $O(Cn)$. The space cost of storing the array and the priority queue is $O(n) + O(n) = O(n)$.
- `EE` (Section 5, Algorithm 2): Let us first consider the initialization cost of parameter values $(\hat{t}, \hat{q})$ (Line 2), using the method discussed in Section 5.4. We first find $E[r_i]$ by scanning the entities in T, in $O(|T|)$ times. Then, we iterate variable $t \in [1, \hat{t}]$, where $\hat{t}$ can be found by Equation 8. For each t, we compute the value of Γ for every $\hat{q} \in [0, \delta, 2\delta, \ldots, (i - 1)\delta, 1]$. Thus, a number $\frac{\hat{t}}{\delta}$ of Γ values will be considered. We then use Lemma 3 to numerically evaluate Γ, using a numerical integration of precision δ (i.e., a sum of $\frac{1}{\delta}$ integration function values). Moreover, each $P_{ne}(p)$ value needs $\hat{t}^2$ time

to complete. Hence, computing Γ requires $O(\frac{\hat{t}^2}{\delta})$ time. The overall initialization cost is $O(|T| + \frac{\hat{t}^3}{\delta^2})$.

For the exploration and exploitation phases (Lines 3-23), notice that each operation takes constant time. Moreover, at most C cleaning operations can be performed. Therefore, this takes $O(C)$ time to complete. Similar to `Random`, an array is used to support the random selection of entities, with a size of $O(n)$.

Algorithm 3 `Random` (AM T, Budget C)

```
1: while C > 0 ∧ ∃j where r_j > 0 do
2:     Randomly select T_j where r_j > 0
3:     if clean(T_j) is successful then
4:         r_j ← r_j − 1
5:     C ← C − 1
```

Algorithm 4 `Greedy` (AM T, Budget C)

```
1: ∀i ∈ [1,n], s_i = 0, t_i = 0, p̂_i = 0
2: while C > 0 ∧ ∃j where r_j > 0 do
3:     Find T_j where r_j > 0 and p̂_j is the highest
4:     if clean(T_j) is successful then
5:         r_j ← r_j − 1
6:         s_j ← s_j + 1
7:     t_j ← t_j + 1
8:     p̂_j ← s_j/t_j
9:     C ← C − 1
```

Algorithm 5 ϵ-`Greedy` (AM T, Budget C, Threshold ϵ)

```
1: ∀i ∈ [1,n], s_i = 0, t_i = 0, p̂_i = 0
2: while C > 0 ∧ ∃j where r_j > 0 do
3:     choice← uniform(0,1)
4:     if choice < ε then
5:         Randomly select T_j where r_j > 0
6:     else
7:         Find T_j where r_j > 0 and p̂_j is the highest
8:     if clean(T_j) is successful then
9:         r_j ← r_j − 1
10:        s_j ← s_j + 1
11:    t_j ← t_j + 1
12:    p̂_j ← s_j/t_j
13:    C ← C − 1
```

C. PROOFS FOR LEMMAS IN SECTION 5

Here we present the correctness proofs of the lemmas used in Section 5.

C.1 Lemma 1 Proof

Since exactly k entities are fully visited, let us assume that there remains some non-zero budget for the $(k+1)$-th entity to be cleaned (the proof can easily handle the case when no budget is available for the $(k+1)$-entity). Without loss of generality, we name these entities $T_1, T_2, \ldots, T_{k+1}$, in ascending order of the time they are chosen to clean. Since entity T_{k+1} is not fully visited, some of $explore(T_{k+1})$ and $exploit(T_{k+1})$ fails to finish before C is exhausted.

Let γ_i be the total number of false values removed from $explore(T_i)$ and $exploit(T_i)$. Also, let χ_i be the sum of cleaning costs required by $explore(T_i)$ and $exploit(T_i)$. Since R_k is the total number of false values removed from these k entities, and C_k is the total cleaning effort invested on them, we have:

$$R_k = \sum_{i=1}^{k} \gamma_i \quad (10)$$

$$C_k = \sum_{i=1}^{k} \chi_i \quad (11)$$

We can then express R and C using Equations 10 and 11, as follows:

$$R = R_k + \gamma_{k+1} \quad (12)$$

$$C = C_k + \chi_{k+1} \quad (13)$$

Recall from Equation 1 that $\xi(\texttt{Framework}) = \frac{E[R]}{C}$. Since C is a constant, $\xi(\texttt{Framework})$ is equal to $E[\frac{R}{C}]$. Moreover, using Equations 12 and 13, we have:

$$\lim_{C\to\infty} \frac{R}{C} = \lim_{k\to\infty} \frac{R_k + \gamma_{k+1}}{C_k + \chi_{k+1}} \quad (14)$$

$$= \lim_{k\to\infty} \frac{R_k/C_k + \gamma_{k+1}/C_k}{1 + \chi_{k+1}/C_k} \quad (15)$$

As discussed, since there is not enough budget for T_{k+1} to be fully visited, γ_{k+1} and χ_{k+1} would be small compared with C_k, which tends to ∞. Hence, both R_k/C_k and χ_{k+1}/C_k vanish, and Equation 15 becomes $\lim_{k\to\infty} \frac{R_k}{C_k}$. Thus,

$$\lim_{C\to\infty} \xi(\texttt{Framework}) = \lim_{k\to\infty} \frac{R_k}{C_k}$$

and the lemma is proved.

C.2 Lemma 2 Proof

Notice that γ_i and χ_i are random variables. Also, the variables γ_i $(i = 1, \ldots, n)$ are independent of each other. Similarly, the random variables χ_i $(i = 1, \ldots, n)$ are also independent of each other. Recall from Equations 10 and 11 that $R_k = \sum_{i=1}^{k} \gamma_i$, and $C_k = \sum_{i=1}^{k} \chi_i$. Then, by using law of large numbers, we have: $\forall \delta > 0$, $\exists K$ such that $\forall k > K$,

$$Prob[|R_k/k - E[\gamma_i]| < \delta] = 1 \quad (16)$$

$$Prob[|C_k/k - E[\chi_i]| < \delta] = 1 \quad (17)$$

where $E[\gamma_i]$ and $E[\chi_i]$ are the expected values of γ_i and χ_i. From Equations 16 and 17, we can state that $\forall \epsilon > 0$, $\exists K$ such that $\forall k > K$,

$$Prob[|R_k/C_k - E[\gamma_i]/E[\chi_i]| < O(\delta)] = 1 \quad (18)$$

Using the definition of limit and expected values, from Equation 18 we obtain the following result:

$$\lim_{k\to\infty} E[R_k/C_k] = E[\gamma_i]/E[\chi_i] \quad (19)$$

By using Lemma 1, we prove the correctness of the lemma.

C.3 Lemma 3 Proof

For **Equation 4**, let the number of false values removed from an entity T_i with sc-probability p be $V(p)$. Then, $E[\gamma_i]$, the expected amount of false values removed, is:

$$E[\gamma_i] = \int_0^1 V(p) \cdot f(p) dp \quad (20)$$

To obtain $V(p)$, we consider two cases for T_i:

- **Case 1:** With probability $P_{ne}(p)$, T_i is only explored, but not exploited. Then T_i has $E_t(p)$ false values removed.
- **Case 2:** With probability $(1 - P_{ne}(p))$, T_i is both explored and exploited. Since this entity has an expected number $E[r_i]$ of false values, using the EE algorithm, $E[r_i]$ false values will be removed.

Hence,

$$V(p) = P_{ne}(p) \cdot E_t(p) + (1 - P_{ne})(p) \cdot E[r_i] \quad (21)$$

The rest of the proof is to obtain $E_t(p)$ and $P_{ne}(p)$. For $E_t(p)$, notice that after exploration, t cleaning operations have been applied on T_i. Since T_i has a sc-probability of p, the expected number of false values that can be removed is bounded by the minimum of tp and $E[r_i]$. Hence, Equation 7 holds.

To obtain $P_{ne}(p)$, we observe that if an entity, which has been explored, is not exploited, the condition that $\hat{p}_i \geq q$ in Step 17 should fail. Since $\hat{p}_i = m/t$ (Step 15), we have:

$$m \leq \lceil tq \rceil - 1 \quad (22)$$

Since m is the number of false values removed, $m \leq E[r_i]$. There are two cases to consider:

- Case 1: $E[r_i] \leq \lceil tq \rceil - 1$. Then, Equation 22 holds, and there is no chance for an exploitation to occur. In other words, $P_{ne}(p) = 1$.
- Case 2: $E[r_i] > \lceil tq \rceil - 1$. Since t cleaning operations is performed during exploration, we can model this as t independent trials, with success probability p. Then, the probability that m cleaning operations fail out of t trials is a binomial distribution: $C_m^t p^m (1-p)^{t-m}$. Moreover, $P_{ne}(p)$ is the probability that Equation 22 occurs, which is equal to $\sum_{m=0}^{\lceil tq \rceil - 1} C_m^t p^m (1-p)^{t-m}$. Hence, Equation 6 holds.

Hence, based on Equations 20 and 21, we conclude that Equation 4 is correct.

To prove the correctness of **Equation 5**, let the amount of cleaning effort spent on an entity T_i with sc-probability p be $S(p)$. Then, $E[\chi_i]$, the expected cleaning cost on each entity is:

$$E[\chi_i] = \int_0^1 S(p) \cdot f(p) dp \quad (23)$$

To obtain $S(p)$, we consider two cases for T_i:

- **Case 1:** With probability $P_{ne}(p)$, T_i is only explored, but not exploited. Then T_i needs a cost of t to be cleaned.
- **Case 2:** With probability $(1 - P_{ne}(p))$, T_i is both explored and exploited. Notice this entity has an expected number $E[r_i]$ of false values. Also, on average, a number $\frac{1}{p}$ of cleaning operations are needed to remove one false value from T_i. Using the EE algorithm, a cost of $\frac{E[r_i]}{p}$ is required to remove $E[r_i]$ false values from T_i.

Hence,

$$S(p) = P_{ne}(p) \cdot t + (1 - P_{ne})(p) \cdot \frac{E[r_i]}{p} \quad (24)$$

By using Equations 6, 23, and 24, Equation 5 is proved.

D. REFERENCES

[1] Jeffrey S Banks and Rangarajan K Sundaram. Switching costs and the gittins index. *Econometrica*, 62(3):687–94, May 1994.

[2] D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, 1985.

[3] Avrim Blum and Yishay Mansour. From external to internal regret. In *In COLT*, pages 621–636, 2005.

[4] Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal Multi-Armed Bandits. In *NIPS*, 2008.

[5] Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *STOC*, pages 334–343, 1997.

[6] J. C. Gittins and D. M. Jones. A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics*, 1974.

[7] Sandeep Pandey, Deepayan Chakrabarti, and Deepak Agarwal. Multi-armed bandit problems with dependent arms. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

[8] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of optimal queuing network control. *MATH. OPER. RES*, 24:293–305, 1999.

[9] Aleksandrs Slivkins and Eli Upfal. Adapting to a changing environment: the brownian restless bandits. In *COLT*, pages 343–354, 2008.

[10] P. Whittle. Arm acquiring bandits. **9**:284–292, 1981.

[11] P. Whittle. Restless bandits: activity allocation in a changing world. In J. Gani, editor, *A celebration of Applied Probability*, Applied Probability Special Volume 25A, pages 287–298. Applied Probability Trust, 1988.