BayesWipe: A Scalable Probabilistic Framework for Improving Data Quality

SUSHOVAN DE, Arizona State University
YUHENG HU, University of Illinois at Chicago
VENKATA VAMSIKRISHNA MEDURI, Arizona State University
YI CHEN, New Jersey Institute of Technology
SUBBARAO KAMBHAMPATI, Arizona State University

Recent efforts in data cleaning of structured data have focused exclusively on problems like data deduplication, record matching, and data standardization; none of the approaches addressing these problems focus on fixing incorrect attribute values in tuples. Correcting values in tuples is typically performed by a minimum cost repair of tuples that violate static constraints like CFDs (which have to be provided by domain experts, or learned from a clean sample of the database). In this paper, we provide a method for correcting individual attribute values in a structured database using a Bayesian generative model and a statistical error model learned from the noisy database directly. We thus avoid the necessity for a domain expert or clean master data. We also show how to efficiently perform consistent query answering using this model over a dirty database, in case write permissions to the database are unavailable. We evaluate our methods over both synthetic and real data.

CCS Concepts: •**Information systems** → **Data cleaning;** •**Mathematics of computing** → **Bayesian networks;** *Probabilistic inference problems; Bayesian computation; Probabilistic algorithms;*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Data quality, statistical data cleaning, offline and online cleaning

ACM Reference Format:
Sushovan De, Yuheng Hu, Venkata Vamsikrishna Meduri, Yi Chen, and Subbarao Kambhampati, 2016. Bayeswipe: a scalable probabilistic framework for improving data quality. *ACM J. Data Inform. Quality* V, N, Article XXXX (September 2016), 30 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Although data cleaning has been a long standing problem, it has become critically important again because of the increased interest in big data and web data. Most of the focus of the work on big data has been on the volume, velocity, or variety of the data; however, an important part of making big data useful is to ensure the veracity of the data. Enterprise data is known to have a typical error rate of 1–5% [Fan and Geerts 2012] (error rates of up to 30% have been observed). This has led to renewed interest in cleaning of big data sources, where manual data cleansing tasks are seen as pro-

This research is supported in part by the ONR grants N00014-13-1-0176, N0014-13-1-0519, ARO grant W911NF-13-1-0023, NSF CAREER award #1322406, a Google Research Grant and an endowment from the Leir Charitable Foundations.
Author's addresses: S. De, Google; Y. Hu, Department of Information and Decision Sciences, University of Illinois at Chicago; Y. Chen, Martin Tuchman School of Management, with a joint appointment at the College of Computing Sciences, New Jersey Institute of Technology; V.V. Meduri and S. Kambhampati, School of Computing, Informatics and Decision Support Engineering; Arizona State University. This work was done when the authors were with the Department of Computer Science & Engineering at Arizona State University. Sushovan De is now at Google.

hibitively expensive and time-consuming [Gray 2013], or the data has been generated by users and cannot be implicitly trusted [Leslie 2010]. Among the various types of big data, the need to efficiently handle large scaled structured data that is rife with inconsistency and incompleteness is also more significant than ever. Indeed, multiple studies, such as [Computing Research Association 2012] emphasize the importance of effective, efficient methods for handling "dirty big data".

Data cleaning is also of paramount importance in web-data scenarios. New and innovative methods of extracting tabular data from the web have shown how informative and significant these sources of data can be [Cafarella et al. 2008]. A critical problem with even the most cutting edge of these techniques [Zhang 2015] is the noise that can get introduced during the extraction. This is where techniques such as the one we describe in this paper can significantly improve the quality of web data.

Table I. A snapshot of car data extracted from cars.com using information extraction techniques

TID	Model	Make	Orig	Size	Engine	Condition
t_1	Civic	Honda	JPN	Mid-size	I4	NEW
t_2	Focus	Ford	USA	Compact	I4	USED
t_3	Civik	Honda	JPN	Mid-size	I4	USED
t_4	Civic	Ford	USA	Compact	I4	USED
t_5		Honda	JPN	Mid-size	I4	NEW
t_6	Accord	Honda	JPN	Full-size	V6	NEW

Most of the current data cleaning techniques are based on deterministic rules, which have a number of problems: Suppose that the user is interested in finding 'Civic' cars from Table I. Traditional data retrieval systems would return tuples t_1 and t_4 for the query, because they are the only ones that are a match for the query term. Thus, they completely miss the fact that t_4 is in fact a dirty tuple — A Ford Focus car mislabeled as a Civic. Additionally, tuples t_3 and t_5 would not be returned as result tuples since they have typos or missing values, although they represent desirable results. The objective of this work is to provide the true result set (t_1, t_3, t_5) to the user.

Although this problem has received significant attention over the years in the traditional database literature, the state-of-the-art approaches fall far short of an effective solution for big data and web data. Traditional methods include outlier detection [Knorr et al. 2000], noise removal [Xiong et al. 2006], entity resolution [Singla and Domingos 2006; Xiong et al. 2006], and imputation [Fellegi and Holt 1976]. Although these methods are efficient in their own scenarios, their dependence on clean master data is a significant drawback.

Specifically, the state-of-the-art approaches (e.g., [Bohannon et al. 2005; Fan et al. 2009; Bertossi et al. 2011]) attempt to clean the data by exploiting the patterns in the data, which they express in the form of CFDs (or Conditional Functional Dependencies). In the motivating example, the fact that the Honda cars have 'JPN' as the origin of the manufacturer would be an example of such a pattern. However, these approaches depend on the availability of a clean data corpus or an external reference table to learn the data quality rules or patterns before fixing the errors in the dirty data. Systems such as ConQuer [Fuxman et al. 2005] depend upon a set of clean constraints provided by the user. Such clean corpora or constraints may be easy to establish in a tightly controlled enterprise environment but are infeasible for web data and big data. One may attempt to learn the data quality rules directly from the noisy data. Unfortunately however, our experimental evaluation in Figure 6(a) from Section 8.2 shows that even small amounts of noise severely impairs the ability to learn useful constraints from the data.

To avoid the dependence on the clean master data, we propose a novel system called BayesWipe [De et al. 2014] that assumes that a statistical process underlies the generation of clean data (which we call the *data source model*) as well as the corruption of data (which we call the *data error model*). The noisy data itself is used to learn the parameters of these generative and error models, eliminating the dependence on the clean master data. Then, by treating the clean value as a latent random variable, BayesWipe leverages these two learned models and automatically infers its value through a Bayesian estimation. The Bayesian inference assigns confidence levels of accuracy to the several possible clean replacement values which we call the candidate set of clean alternatives.

Deterministic cleaning is ineffective as compared to BayesWipe for the following reasons:

— Most of the rule based methods like (C)FDs cannot generate rules in the presence of even a single violating tuple in the master data which is why they can learn close to zero patterns (rules) from noisy data.
— AFDs (Approximate Functional Dependencies) can learn rules from noisy data but they can suffer from other semantic inconsistencies when interactions are allowed between rules. For example, applying transitivity between rules can lead to circular reasoning and in some cases, can exaggerate the truth by overcounting the evidence (Section 14.7.1 of [Russell and Norvig 2010]). This results in the need to carefully doctoring the rules such that there is no interference among themselves to avoid such inconsistencies.
— BayesWipe, in contrast, learns the generative model from the majority of the data which makes it robust to the noise, thus enabling it to build its data source model from dirty data, while also avoiding the problem of semantic inconsistencies. We also present the crowdsourcing based experiments in Table IV from Section 8.2 to show that the candidate clean tuples suggested by the generative model built upon the dirty data are indeed accurate and are consistent with the clean replacements picked by the human participants.

We designed BayesWipe so that it can be used in two different modes: a traditional *offline cleaning* mode, and a novel *online query processing* mode. The offline cleaning mode of BayesWipe follows the classical data cleaning model, where the entire database is accessible and can be cleaned *in situ*. This mode is particularly useful when one has complete control over the data, and a one-time cleaning of the data is needed. Data warehousing scenarios such as the data crawled from the web, or aggregated from various noisy sources can be effectively cleaned in this mode. One of the features of the offline mode of BayesWipe is that a PDB (probabilistic database) can be generated as a result of the data cleaning. The cleaned data can be stored either in a deterministic database, or in a probabilistic database. If a probabilistic database is chosen as the output mode, BayesWipe stores not only the clean version of the tuple it believes to be most likely correct one, but the entire distribution over the possible clean tuples available in the candidate set. The choice of a probabilistic output mode for the cleaned tuples is most useful for those scenarios where recall is very important for further data processing on the cleaned tuples.

Probabilistic databases are complex and unintuitive, because each single input tuple is mapped into a distribution over resulting clean alternatives. We show how the top-k results can be retrieved from a PDB while displaying the clean data that is comprehensible to the user.

The online query processing mode of BayesWipe is motivated by web data scenarios where it is impractical to create a local copy of the data and clean it offline, either due to large size, high frequency of change, or access restrictions. In such cases, the best way to obtain clean answers is to clean the resultset as we retrieve it, which also provides us the opportunity of improving the efficiency of the system, since we can now ignore entire portions of the database which are likely to be unclean or irrelevant to the top-k results. BayesWipe uses a *query rewriting system* that enables it to efficiently retrieve only those tuples that are important to the top-k result set. This rewriting approach is inspired by, and is a significant extension of our earlier work on QPIAD system for handling data incompleteness [Wolf et al. 2009]. In big data scenarios, clean master data is rarely available, and write access is either unavailable, or undesirable due to the efficiency and indexing concerns. The online mode is particularly suited to get clean results in such cases.

We implement BayesWipe in a Map-Reduce architecture, so that we can run it very quickly for massive datasets. The architecture for parallelizing BayesWipe is explained more fully in Sec 7. In short, there is a two-stage map-reduce architecture, where in the first stage, the dirty tuples are routed to a set of reducer nodes which hold the relevant candidate clean tuples for them. In the second stage, the resulting candidate clean tuples along with their scores are collated, and the best replacement tuple is selected from them.

To summarize our contributions, we:

— Propose that data cleaning should be done using a principled, probabilistic approach.
— Develop a novel algorithm following those principles, which uses a Bayes network as the generative model and maximum entropy as the error model of the data.
— Develop novel query rewriting techniques so that this algorithm can also be used in a big data scenario.
— Develop a parallelized version of this algorithm using map-reduce framework.
— Empirically evaluate the performance of our algorithm using both controlled and real datasets.

The rest of the paper is organized as follows. We begin by discussing the related work and then describe the architecture of BayesWipe in the next section, where we also present the overall algorithm. Section 4 describes the learning phase of BayesWipe, where we find the generative and error models. Section 5 describes the offline cleaning mode, and the next section details the query rewriting and online data processing. We describe the parallelized version of BayesWipe in Section 7 and the results of our empirical evaluation in Section 8, and then conclude by summarizing our contributions. Further details about BayesWipe can be found in the thesis [De 2014].

2. RELATED WORK

Much of the work in data cleaning focused on deterministic dependency relations such as FDs (Functional Dependencies), CFDs (Conditional Functional Dependencies), AFDs (Approximate Functional Dependencies) and INDs (Inclusion Dependencies). Bohannon *et al.* proposed using CFDs to clean the data [Bohannon et al. 2007; Fan et al. 2008]. Indeed, CFDs are very effective in cleaning the data. However, the precision and recall of cleaning the data with CFDs completely depends on the quality of the set of dependencies used for cleaning. As our experiments show, learning CFDs from dirty data produces very unsatisfactory results. In order for CFD-based methods to perform well, they need to be learned from a clean sample of the database [Fan et al. 2009] which must be large enough to be representative of all the patterns in the data. Finding such a large corpus of clean master data is a non-trivial problem, and is infeasible in all but the most controlled of environments (like a corporation with high quality data).

A recent variant on the deterministic dependency based cleaning by J.Wang *et al.* [Wang and Tang 2014] proposes using fixing rules containing negative and positive patterns which indicate the possible errors and the corresponding clean replacements respectively for an attribute. However, there can be several ways in which a tuple can go wrong and the detection of the positive pattern requires clean master data.

BayesWipe, on the other hand, uses an error model to detect the errors automatically and clean them in the absence of clean master data. Recent work by J.Wang *et al.* [Wang et al. 2014] plugs in one of the rule based cleaning techniques to clean a sample of the data and use it as a guideline to clean the entire data. It is important to note that this method only caters to aggregate numerical queries whereas the online mode of BayesWipe supports selecting the actual clean values and can be easily extended to support all kinds of queries.

Although it is possible to ameliorate some of the difficulties of CFD/AFD methods by considering approximate versions of them, the work in the uncertainty in AI community demonstrated the semantic pitfalls of handling uncertainty in this way. In particular, approximate versions of CFDs/AFDs considered in works such as [Golab et al. 2008; Cormode et al. 2009] are similar to the certainty factors approaches for handling uncertainty that were popular in the heyday of expert systems, but whose semantic inconsistencies are by now well-established (see, for example, Section 14.7.1 of [Russell and Norvig 2010]). Because of this, in this paper we focus on a more systematic probabilistic approach.

Even if a curated set of integrity constraints are provided, existing methods do not use a probabilistically principled method of choosing a candidate correction. They resort to either heuristic based methods, finding an approximate algorithm for the least-cost repair of the database [Arenas et al. 1999; Bohannon et al. 2005; Cong et al. 2007]; using a human-guided repair [Yakout et al. 2011], or sampling from a space of possible repairs [Beskales et al. 2013b]. There has been work that attempts to guarantee a correct repair of the database [Fan et al. 2010], but they can only provide guarantees for corrections of those tuples that are supported by data from a perfectly clean master database. Recently, [Beskales et al. 2013a] have shown how the relative trust one places on the constraints and the data itself plays into the choice of cleaning tuples. A Bayesian source model of data was used by [Dong et al. 2009], but was limited in scope to figuring out the evolution over time of the data value.

Kubica and Moore [2003] describe an method for data cleaning with a data, noise and corruption model. The corruption model determines which values in the database are corrupt, and the noise model determines what values they are replaced with. Thus fundamentally, their noise model is different; instead of modeling the corruption itself like $P(T|T^*)$, they learn a generative noise: $P(T)_{T\text{ is corrupt}}$. At every iteration of their EM algorithm, they split the data into two parts: a set of values presumably correct, which they use to learn the clean model, and a set of values presumably corrupt, which they use to learn the noise model. Certain well-known families of probability distributions (such as Gaussian and uniform distribution) are used for the models. During the "learning" phase, they iteratively refine the parameters of these models. On the other hand, we learn a sophisticated Bayesian network directly from the dirty data, and profit from a comprehensive error model that mirrors common real-world errors. This also allows us to support online querying over remote databases, something that LENS cannot support.

Mayfield *et al.* [2009] describe a statistical data cleaning application where approximate Bayesian inference is used as the underlying model for inferring clean values. However, their work differs from ours in a number of important ways: first, they focus on a domain where *Shrinkage* by convolution is possible. This means that it is possible to use a rule (like $death\_age = death\_year - birth\_year$) to vastly reduce the size of the domain and learn CPDs from the examples. Such a method would not be applicable to categorical data like makes and models of cars. Secondly, their approach does not use an error model, and instead fixes values that are missing, or readily identified as outliers. By using a probabilistic error model, BayesWipe is able to treat every value in every tuple as possibly erroneous, and compute the probabilistically cleanest correction.

Recent work has also focused on the metrics to be used to evaluate the data cleaning techniques [Dasu and Loh 2012]. In this work, we focus on evaluating our method against the ground truth (when the ground truth is known), and user studies (when the ground truth is not known).

While BayesWipe uses crowdsourcing to evaluate the accuracy of the proposed clean tuple alternatives for the experiments on real world datasets, there are other systems that try to use the crowd for cleaning the data itself. X.Chu *et al.* [Chu et al. 2015] clean the database tuples by discovering patterns that overlap with KB(Knowledge Base)s like Yago and validating the top-k candidates using the crowd. J.Wang *et al.* [Wang et al. 2012] perform entity resolution (which is to identify several values corresponding to the same entity value) using crowdsourcing. They reduce the complexity of the number of HIT (Human Intelligence Tasks) generated by clustering them into several bins so that a set of pairs can be resolved at a time as against evaluating one pair at a time. Y.Zheng *et al.* [Zheng et al. 2015] pick a set of k questions to be included in the HITs for the human workers out of a total set of n questions using estimates on the expected increase in the answer quality by assigning those questions to the crowd. Crowdsourcing to perform data cleaning may be infeasible in the context of Big Data cleaning targeted by BayesWipe . However, suggestions from the crowd can be used to pre-clean a sample of the dirty data from which BayesWipe learns the Bayes network.

The query rewriting part of this work is inspired by the QPIAD system [Wolf et al. 2009], but significantly improves upon it. QPIAD performed query rewriting over incomplete databases using AFDs, and only cleaned data with null values, not *wrong* values. Arenas *et al.* show [Arenas et al. 1999] a method to generate rewritten queries to obtain clean tuples from an inconsistent database. However, the query rewriting algorithm in that paper is driven by the deterministic integrity dependencies, and not the generative or error model. Since their system requires a set of curated deterministic dependencies, it is not directly applicable to the problem solved in this work. Furthermore, due to the use of Bayes networks to build the generative model, BayesWipe is able to incorporate richer types of dependencies.

3. BAYESWIPE OVERVIEW

BayesWipe views the data cleaning problem as a statistical inference problem over tuples of categorical data. In this paper, we support "select" queries on a single table D[1] containing the dirty data. (An example of a select query represented as SQL might be, "select * from db where make='honda' and model='civic' ".) We do not support relational data that spans multiple tables, however if it is possible to extract a view of the relational data into a set of tuples, then BayesWipe can be operated on it. The current system can be extended to support other relational operators like joins and aggregates but that would more be an engineering exercise and is beyond the scope of a single paper. This extension to support selection queries involving further database operators can also handle the connectivity among web data when it can be effectively

[1]A few words are in order about the relational model we use. Although the linked structure of the web might suggest a graphical model, relational model does apply quite widely for the web data. Structured data on the web may appear explicitly in the form of tables, but they may also form the underlying data of millions of web-pages, that can be scraped one or more tuples at a time. The extraction of meaningful relational data from the web has been of significant research interest in the recent past [Zhang 2015; Weninger and Han 2013; Zhang et al. 2012; Hoffmann et al. 2011; Hoffmann et al. 2010; Cafarella et al. 2008]. In fact, spin-off companies such as *Lattice* (https://lattice.io/) are especially geared towards deriving a structured representation of unstructured data. In addition to that, the popular representation of web data using RDF triples is a proof enough to indicate the prevalence of structure on the web. By aiming BayesWipe at structured data, we are providing a method to significantly improve the data quality of the web.

represented using foreign keys over which the application of natural joins is straightforward. However, the current support offered to queries containing such operators is to first execute them and store the materialized view in a table upon which BayesWipe can be applied.

Let $\mathcal{D} = \{T_1, ..., T_n\}$ be the input relation (like Table I) which contains a number of corruptions. $T_i \in \mathcal{D}$ is a tuple with m attributes $\{A_1, ..., A_m\}$ which may have one or more corruptions in its attribute values. Given a candidate replacement set $\mathcal{C}$ for a possibly corrupted tuple T in $\mathcal{D}$, we can clean the database by replacing T with the candidate clean tuple $T^* \in \mathcal{C}$ that has the maximum $\mathbf{Pr}(T^*|T)$. Using Bayes rule (and dropping the common denominator), we can rewrite this to

$$T^*_{best} = \arg\max[\mathbf{Pr}(T|T^*)\mathbf{Pr}(T^*)] \tag{1}$$

By replacing T with T^*_{best}, we get a deterministic database. If we wish to create a PDB (probabilistic database), we do not take an $\arg\max$ over the $\mathbf{Pr}(T^*|T)$, instead we store the entire distribution over the T^* in the resulting PDB.

It is important to note that the candidate replacement set $\mathcal{C}$ is actually derived from a sample of the dirty data using a generative model (explained in Section 4.1) as we do not assume the availability of clean master data.

For online query processing we rewrite the user query Q^* into Q, and find the relevance score of a tuple T as

$$Score(T) = \sum_{T^* \in \mathcal{C}} \underbrace{\mathbf{Pr}(T^*)}_{\text{source model}} \underbrace{\mathbf{Pr}(T|T^*)}_{\text{error model}} \underbrace{R(T^*|Q^*)}_{\text{relevance}} \tag{2}$$

In this work, we used a binary relevance model, where R is 1 if T^* is relevant to the user's query Q^* and 0 otherwise. A tuple T^* is deemed relevant to Q^* if it satisfies the query constraints and can participate in the exact answer set of the query Q^* executed on the relational table D. Note that R is the relevance of the query Q^* to the candidate clean tuple T^* and not the observed tuple T. An observed tuple T achieves a high relevance $Score(T)$ if each of its candidate replacement tuples satisfy the query Q^* and have a high posterior probability $\mathbf{Pr}(T^*|T)$ of replacing T. This allows the query rewriting phase of BayesWipe to rewrite a user query Q^* into Q such that the execution of the rewritten query Q retrieves tuples with the highest relevance $Score(.)$ with respect to the original query Q^*. The motivation behind rewriting Q^* to Q is to fetch clean query answers which are otherwise missed by executing the original query and is explained in better detail in Section 6. The retrieval of the top-k tuples with a high relevance score is to achieve the non-lossy effect of using a PDB without explicitly rectifying the entire database.

Architecture:

Figure 1 shows the system architecture for BayesWipe. During the model learning phase (Section 4), we first obtain a sample database by sending some queries to the database. On this sample data, we learn the generative model of the data as a Bayes network (Section 4.1). In parallel, we define and learn an error model which incorporates common kinds of errors (Section 4.2). We also create an index to quickly propose candidate T^*s.

We can then choose to do either offline cleaning (Section 5) or online query processing (Section 6), as per the scenario. In the offline cleaning mode, we iterate over all the tuples in the database and clean them one by one. We can choose whether to store the resulting cleaned tuple in a deterministic database (where we store only the T^* with the maximum posterior probability) or probabilistic database (where we store the entire distribution over the T^*). In the online query processing mode, we obtain a query from the user, and do query rewriting in order to find a set of queries that are

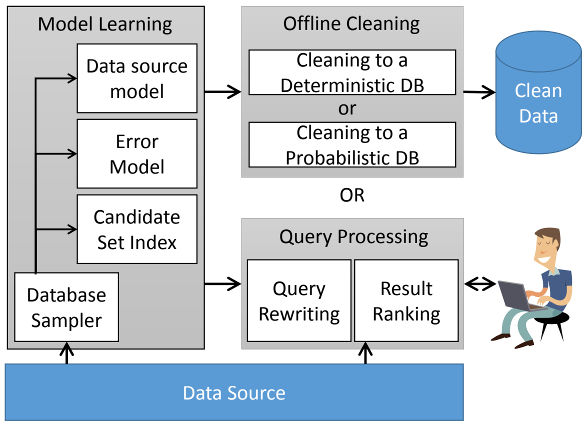
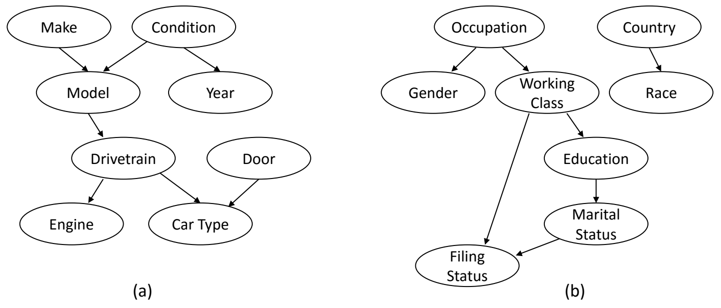


Fig. 1. The architecture of BayesWipe. Our framework learns both data source model and error model from the raw data during the model learning phase. It can perform offline cleaning or query processing to provide clean data.

likely to retrieve a set of highly relevant tuples. We execute these queries and re-rank the results, and then display them.

4. MODEL LEARNING



Tuple Id	Make	Model	Condition	Year	Drivetrain	Engine	Car Type	Doors
T*	Honda	Civic	NEW	2015	FWD	I6	Hybrid	4

Pr(T*) = **Pr**(Make = Honda) **Pr**(Condition = NEW) **Pr**(Model = Civic | Make = Honda, Condition = NEW)
Pr (Year = 2015 | Condition = NEW) **Pr**(Drivetrain = FWD | Model = Civic)
Pr(Door = 4) **Pr**(Engine = I6 | Drivetrain = FWD)
Pr(Car Type = Hybrid | Drivetrain = FWD, Door = 4)

Fig. 2. The learned Bayes networks

This section details the process by which we estimate the components of Equation 2: the data source model $\mathbf{Pr}(T^*)$ and the error model $\mathbf{Pr}(T|T^*)$.

4.1. Data Source Model

Our data source model is a generative model built from a sample of the dirty data, $\mathcal{D}$. It computes a statistical measure in the form of a joint prior probability, $\mathbf{Pr}(T^*)$, for each tuple T^* in $\mathcal{D}$ which is is used along with the error likelihood from Section 4.2 to estimate its effectiveness in cleaning a dirty tuple T.

The data that we work with can have dependencies among various attributes For example, a car's *engine* depends on its *make*, since cars made by the same manufacturers tend to use the engines of the same type. Although the engine does not exclusively determine the make of the car, we rely upon additional information in the form of probabilities which represent the strength of such dependencies among attributes. It helps us in disregarding manufacturers who do not use common engines like I4 and V6 as their joint probabilities fall very low. Therefore, we represent the data source model as a Bayes network, since it naturally captures relationships between the attributes via structure learning and infers probability distributions over values of the input tuples.

Constructing a Bayes network over $\mathcal{D}$ requires two steps: first, the induction of the graph structure of the network, which encodes the conditional independences between the m attributes of $\mathcal{D}$'s schema; and second, the estimation of the parameters of the resulting network. The resulting model allows us to compute probability distributions over an arbitrary input tuple T.

Whenever the underlying patterns in the source database changes, we have to learn the structure and parameters of the Bayes network again. In our scenario, we observed that the structure of a Bayes network of a given dataset remains constant with small perturbations, but the CPTs (Conditional Probability Tables) change more frequently. As a result, we spend a larger amount of time learning the structure of the network with a slower, but more accurate tool, Banjo [Hartemink. 2005]. Figure 2 shows automatically learned structures for two data domains. The learned structure seems to be intuitively correct, since the nodes that are connected (for example, 'country' and 'race' in Figure 2(b)) are expected to be highly correlated[2].

Then, given a learned graphical structure $\mathcal{G}$ of $\mathcal{D}$, we can estimate the CPTs that parameterize each node in $\mathcal{G}$ using a faster package called Infer.NET [Minka et al. 2010]. This process of inferring the parameters is run offline, but more frequently than the structure learning.

Once the Bayesian network is constructed, we can infer the joint distributions for an arbitrary tuple T^*. This distribution can be decomposed to the multiplication of several conditional distributions of the sets of random variables, conditioned on their parent nodes depending on $\mathcal{G}$. Figure 2(a) has an example tuple T^* for which the attribute correlations are derived from the Bayes network structure and the joint probability is computed from the conditional probabilities available in the accompanying CPT. Since the Bayes network is learnt from the dirty data, each source model tuple T^* is implicitly weighted by its prior probability $\mathbf{Pr}(T^*)$ (and its error likelihood based on the distance from the observed dirty tuple T, explained in Section 4.2) which can act as an estimate of the confidence with which the Bayes network supports the generation of this tuple as a clean candidate. In other words, a tuple from the data sample with a low prior probability (and a low error likelihood) is unlikely to act as a clean replacement for a dirty tuple.

4.2. Error Model

Having described the data source model, we now turn to the estimation of the error model $\mathbf{Pr}(T|T^*)$ from the noisy data. There are many types of errors that can occur in the data. We focus on some of the most common types of errors that occur in the data that is manually entered by naïve users: typos, deletions, and substitution of one word with another. However, we are not dealing with errors that occur in the entry of long-form text, such as common spelling mistakes with everyday English words, or parsing and unit conversion errors. We particularly focus on categorical, tabular data.

[2]Note that the direction of the arrow in a Bayes network does not necessarily determine causality, see Chapter 14 from Russell and Norvig [Russell and Norvig 2010].

Other types of common errors such as those due to encoding problems, or formatting issues can be tackled by straightforward applications of rule-based scripts or ETL tools. Raman and Hellerstein [2001] mention the prevalent use of ETL tools like Data Junction and DataStage to transform data into a unified encoding format. They use a GUI-aided interactive transformation step to quickly build a custom script to fix the encoding errors. Kandel *et al.* also propose the use of richer visual interfaces for transformation techniques to address the encoding errors in columns such as 'Date' [Kandel et al. 2011]. We will assume in this work that such pre-processing has already been done on the data if necessary, and consider those types of errors to be out of scope for this work.

The error model we currently use can be further extended to include more complex errors. Nevertheless we show in Section 8 two sets of experiments: one in which we synthetically introduce the errors as per the error model and another in which the naturally occurring data is considered without a controlled error model. The percentage of the tuples cleaned in the latter case as shown in Table IV shows that our error model is indeed relevant and it is further corroborated by the opinion of the crowd using which we evaluate our cleaning performance. We also make two assumptions: First, that the same type of error does not occur so frequently that it is replicated more often than the correct value. In such a (albeit extremely unlikely) case, the model learner will learn the erroneous value as the right one. Our second assumption is that an error in the value of one attribute does not affect the errors in the values of other attributes. This is a reasonable assumption to make, since we are allowing the data itself to have dependencies between attributes, while only constraining the error process to be independent across attributes. With these assumptions, we are able to come up with a simple and efficient error model, where we combine the three types of errors using a maximum entropy model.

Given a set of clean candidate tuples $\mathcal{C}$ where $T^* \in \mathcal{C}$, our error model $\mathbf{Pr}(T|T^*)$ essentially measures how clean T is, or in other words, how similar T is to T^*.

Edit distance similarity: This similarity measure is used to detect spelling errors. Edit distance between two strings T_{A_i} and $T^*_{A_i}$ is defined as the minimum cost of edit operations applied to a dirty tuple T_{A_i} in order to transform it to a clean tuple, $T^*_{A_i}$. Edit operations include character-level copy, insert, delete and substitute. The cost for each operation can be modified as required; in this paper we use the Levenshtein distance, which uses a uniform cost function. This gives us a distance, which we then convert to a probability using [Ristad and Yianilos 1998]:

$$f_{ed}(T_{A_i}, T^*_{A_i}) = \exp\{-cost_{ed}(T_{A_i}, T^*_{A_i})\} \tag{3}$$

Distributional Similarity Feature: This similarity measure is used to detect both substitution and omission errors (null input values representing omissions). Looking at each attribute in isolation is not enough to fix these errors. We propose a context-based similarity measure called Distributional similarity (f_{ds}), which is based on the probability of replacing one value with another under a similar context [Li et al. 2006]. Formally, for each string T_{A_i} and $T^*_{A_i}$, we have:

$$f_{ds}(T_{A_i}, T^*_{A_i}) = \sum_{c \in C(T_{A_i}, T^*_{A_i})} \frac{\mathbf{Pr}(c|T^*_{A_i})\mathbf{Pr}(c|T_{A_i})\mathbf{Pr}(T_{A_i})}{\mathbf{Pr}(c)} \tag{4}$$

where $C(T_{A_i}, T^*_{A_i})$ is the context of a tuple attribute value, which is a set of attribute values that co-occur with both T_{A_i} and $T^*_{A_i}$. $\mathbf{Pr}(c|T^*_{A_i}) = (\#(c, T^*_{A_i}) + \mu)/\#(T^*_{A_i})$ is the probability that a context value c appears given the clean attribute $T^*_{A_i}$ in the sample database. Similarly, $P(T_{A_i}) = \#(T_{A_i})/\#tuples$ is the probability that a dirty attribute value appears in the sample database. We calculate $\mathbf{Pr}(c|T_{A_i})$ and $\mathbf{Pr}(T_{A_i})$ in the same way. To avoid zero estimates for attribute values that do not appear in the database sample, we use Laplace smoothing factor μ.

Unified error model: In practice, we do not know beforehand which kind of error has occurred for a particular attribute; we need a unified error model which can accommodate all three types of errors (and be flexible enough to accommodate more errors when necessary). For this purpose, we use the well-known maximum entropy framework [Berger et al. 1996] to leverage both the similarity measures, (Edit distance f_{ed} and distributional similarity f_{ds}). For each attribute of the input tuple T and T^*, we have the unified error model $\mathbf{Pr}(T|T^*)$ given by:

$$\frac{1}{Z}\exp\left\{\alpha\sum_{i=1}^{m} f_{ed}(T_{A_i}, T^*_{A_i}) + \beta\sum_{i=1}^{m} f_{ds}(T_{A_i}, T^*_{A_i})\right\} \tag{5}$$

where α and β are the weight of each feature, m is the number of attributes in the tuple. The normalization factor is $Z = \sum_{T^*}\exp\{\sum_i \lambda_i f_i(T^*, T)\}$, where λ_i is the weight of the i-th feature. We explain how to set the values of α and β in Section 5.1 and experiment with it in Figure 6(c).

4.3. Finding the Candidate Set

The set of candidate tuples, $\mathcal{C}(T)$ for a given tuple T are the possible replacement tuples that the system considers as possible corrections to T. The larger the set $\mathcal{C}$ is, the longer it will take for the system to perform the cleaning. If $\mathcal{C}$ contains many unclean tuples, then the system will waste time scoring tuples that are not clean to begin with. It should be noted that we also add the tuple itself, T, to the set of the candidate tuples.

An efficient approach to finding a reasonably clean $\mathcal{C}(T)$ is to consider the set of all the tuples in the sample database that differ from T in not more than j attributes. In order to find $\mathcal{C}(T)$ that satisfies this, conceptually, we have to iterate over every tuple t in the sample database D, comparing it to the tuple T and checking how many attributes it differs in. This operation can take $\mathcal{O}(n)$ time, where n is the number of tuples in the sample database. Even with $j = 3$, the naïve approach of constructing $\mathcal{C}$ from the sample database directly is too time consuming, since it requires one to go through the sample database in its entirety once for every result tuple encountered. To make this process faster, we create indices over $(j + 1)$ attributes because searching through indices reduces the number of comparisons required to compute $\mathcal{C}(T)$. If any candidate tuple T^* differs from T in less than or equal to j attributes, then it will be present in at least one of the indices, since we created $j + 1$ of them (pigeon hole principle). These $j + 1$ indices are created over those attributes that have the highest cardinalities, such as Make and Model (as opposed to attributes like Condition and Doors which can take only a few values). This ensures that the set of tuples returned from the index would be small in number.

For every possibly dirty tuple T in the database, we go over each such index and find all the tuples that match the corresponding attribute. The union of all these tuples is then examined and the candidate set $\mathcal{C}$ is constructed by keeping only those tuples from this union set that do not differ from T in more than j attributes. Thus we can be sure that by using this method, we have obtained the entire set $\mathcal{C}$ [3].

[3]There is a small possibility that the true tuple T^* is not in the sample database at all. This probability can be reduced by choosing a larger sample set. In future work, we will expand the strategy of generating $\mathcal{C}$ to include all possible k-repairs of a tuple.

5. OFFLINE CLEANING

In Algorithm 1, we present the offline mode of BayesWipe. We show how we iterate over all the tuples in the dirty database, D and replace them with cleaned tuples.

ALGORITHM 1: The algorithm for offline data cleaning

```
Input: $D$, the dirty dataset.
$BN \leftarrow$ Learn Bayes Network ($D$)
foreach Tuple $T \in D$ do
    $\mathcal{C} \leftarrow$ Find Candidate Replacements ($T$)
    foreach Candidate $T^* \in \mathcal{C}$ do
        $P(T^*) \leftarrow$ Find Joint Probability ($T^*, BN$)
        $P(T|T^*) \leftarrow$ Error Model ($T, T^*$)
    end
    $T \leftarrow \arg\max_{T^* \in \mathcal{C}} P(T^*)P(T|T^*)$
end
```

5.1. Cleaning to a Deterministic Database

In order to clean the data *in situ*, we first use the techniques of the previous section to learn the data source model, the error model and create the index. Then, we iterate over all the tuples in the database and use Equation 1 to find the T^* with the best score. We then replace the tuple with that T^*, thus creating a deterministic database using the offline mode of BayesWipe.

Computing $\mathbf{Pr}(T^*)\mathbf{Pr}(T|T^*)$ is very fast. Even though we do a Bayesian inference for $\mathbf{Pr}(T^*)$, the tuple has all the values specified, so the inference ends up being a simple multiplication over the CPTs of the Bayes network, and is very cheap. $\mathbf{Pr}(T|T^*)$ involves simple edit distance and distributional similarity calculations all of which involve simple arithmetic operations and lookups devoid of Bayesian inference.

Recall from Section 4.2 that there are parameters in the error model called α and β, which need to be set. Interestingly, in addition to controlling the relative weight given to the various features in the error model, these parameters can be used to control overcorrection by the system.

Overcorrection: Any data cleaning system is vulnerable to overcorrection, where a legitimate tuple is modified by the system to an unclean value. Overcorrection can have many causes. In a traditional, deterministic system, overcorrection can be caused by erroneous rules learned from infrequent data. For example, certain makes of cars are all owned by the same conglomerate (GM owns Chevrolet). In a misguided attempt to simplify their inventory, a car salesman might list all the cars under the name of the conglomerate. This may provide enough support to learn the wrong rule (Malibu $\rightarrow$ GM).

Typically, once an erroneous rule has been learned, there is no way to correct it or ignore it without a lot of oversight from domain experts. However, BayesWipe provides a way to regulate the amount of overcorrection in the system with the help of a 'degree of change' parameter. Without loss of generality, we can rewrite Equation 5 to the following:

$$\mathbf{Pr}(T|T^*) = \frac{1}{Z}\exp\Bigg\{\gamma\Big(\delta\sum_{i=1}^{m} f_{ed}(T_{A_i}, T^*_{A_i}) + (1-\delta)\sum_{i=1}^{m} f_{ds}(T_{A_i}, T^*_{A_i})\Big)\Bigg\}$$

Since we are only interested in their relative weights, the parameters α and β have been replaced by δ and $(1-\delta)$ with the help of a normalization constant, γ. This parameter, γ, can be used to modify the degree of variation in $\mathbf{Pr}(T|T^*)$. High values of γ imply that small differences in T and T^* cause a larger difference in the value of $\mathbf{Pr}(T|T^*)$, causing the system to give higher scores to the original tuple (compared to a modified tuple). Hence, γ is the overcorrection parameter which regulates the extent to which the original tuple T is modified, thus preserving the original tuple. δ is the relative weight assigned to the edit distance similarity as against $(1-\delta)$ assigned to the distributional similarity.

Example: Consider the following fragment from the database. The first tuple is a very frequent tuple in the database, the second one is an erroneous tuple, and the third tuple is an infrequent, correct tuple. The 'true' correction of the second tuple is the third tuple. The $\mathbf{Pr}(T^*)$ values shown reflect the values that the data source model might predict for them, roughly based on the frequency with which they occur in the source data.

Id	Make	Model	Type	Engine	Condition	$P(T^*)$
1	Honda	Civic	Sedan	I4	New	0.400
2	Honda	750	Sedan	V8	New	0.001
3	BMW	750	Sedan	V8	New	0.005

A proper data cleaning system will correct tuple 2 to tuple 3, and not modify any of the others. However, if incorrect rules (for example, 750 $\rightarrow$ Honda) were learned, there could be overcorrection, where tuple 3 is modified to tuple 2.

On the other hand, BayesWipe handles this situation based on the value of γ. Looking at tuple 3 (which is a clean tuple), suppose the candidate replacement tuples for it are also tuples 1, 2 and 3. In that case, the situation may look like the following:

		low γ		high γ	
Candidate	$P(T^*)$	$P(T\|T^*)$	score	$P(T\|T^*)$	score
1	0.400	0.02	**0.0080**	0.002	0.00080
2	0.001	0.30	0.0003	0.030	0.00003
3	0.005	1.00	0.0050	1.000	**0.00500**

As we can see, if we choose a low value of γ, the candidate with the highest score is tuple 1, which means an overcorrection will occur. However, with higher γ, the candidate with the highest score is tuple 3 itself, which means the tuple will not be modified, and overcorrection will not occur. On the other hand, if we set γ too high, then even legitimately dirty tuples like tuple 2 will not get changed, thus the number of actual corrections will also be lower.

To make full use of this capability of regulating overcorrection, we need to be able to set the value of γ appropriately. In the absence of a training dataset (for which the ground truth is known), we can only estimate the best γ approximately. We do this by finding a value of γ for which the percentage of tuples modified by the system is equal to the expected percentage of noise in the dataset.

5.2. Cleaning to a Probabilistic Database

We note that many data cleaning approaches — including the one we described in the previous sections — come up with multiple alternatives for the clean version for any given tuple, and evaluate their confidence in each of the alternatives. For example, if a tuple is observed as 'Honda, Corolla', two correct alternatives for that tuple might be 'Honda, Civic' and 'Toyota, Corolla'. In such cases, where the choice of the clean tuple is not an obvious one, picking the most-likely option may lead to the wrong answer. Additionally, if we intend to do further processing on the results, such as perform aggregate queries, join with other tables, or transfer the data to someone else for processing, then storing the most likely outcome is lossy.

A better approach (also suggested by others [Computing Research Association 2012]) is to store all of the alternative clean tuples along with their confidence values. Doing this, however, means that the resulting database will be a probabilistic database (PDB), even when the source database is deterministic.

It is not clear upfront whether PDB-based cleaning will have advantages over cleaning to a deterministic database. On the positive side, using a PDB helps reduce loss of information arising from discarding all alternatives to tuples that did not have the maximum confidence. On the negative side, PDB-based cleaning increases the query processing cost (as querying PDBs are harder than querying deterministic databases [Dalvi and Suciu 2004]).

Another challenge is one of presentation: users usually assume that they are dealing with a deterministic source of data, and presenting all alternatives to them can be overwhelming to them. In this section, and in the associated experiments, we investigate the potential advantages to using the BayesWipe system and storing the resulting cleaned data in a probabilistic database. For our experiments, we used Mystiq [Boulos et al. 2005], a prototype probabilistic database system from University of Washington, as the substrate. In order to create a probabilistic database from the corrections of the input data, we follow the offline cleaning procedure described previously in Section 4. Instead of storing the most likely T^*, we store all the T^*s along with their $P(T^*|T)$ values. When evaluating the performance of the probabilistic database, we used simple select queries on the resulting database. Since representing the results of a probabilistic database to the user is a complex task, in this paper we focus on showing the XOR representation of the tuple alternatives to the user. The rationale for our decision is that in a used car scenario, the user will be provided with a URL link to the car through the clickable tuple id and the several alternative clean values for the dirty attributes are shown within the single tuple returned to the user. As a result, the form of our output is a tuple-disjoint independent database [Suciu and Dalvi 2005]. This can be better explained with an example:

Table II. Cleaned probabilistic database

TID	Model	Make	Orig.	Size	Eng.	Cond.	P
t_1	Civic	Honda	JPN	Mid-size	I4	NEW	0.6
	Civic	Honda	JPN	Compact	V6	NEW	0.4
...							
t_3	Civic	Honda	JPN	Mid-size	I4	USED	0.9
	Civik	Honda	JPN	Mid-size	I4	USED	0.1

Example: Suppose we clean our running example of Table I. We will obtain a tuple-disjoint independent[4] probabilistic database [Suciu and Dalvi 2005]; a fragment of which is shown in Table II. Each original input tuple (t_1, t_3), has been cleaned, and their alternatives are stored along with the computed confidence values for the alternatives (0.6 and 0.4 for t_1, in this example). Suppose the user issues a query Model = Civic. Both options of tuple t_1 of the probabilistic database satisfy the constraints of the query. Since there are two valid alternatives to tuple t_1 in the result with probabilities

[4]A tuple-disjoint independent probabilistic database is one where every tuple, identified by its primary key, is independent of all other tuples. Each tuple is, however, allowed to have multiple alternatives with associated probabilities. In a tuple-independent database, each tuple has a single probability, which is the probability of that tuple existing.

0.6 and 0.4, in order to get a single tuple representation, the matching attributes in the alternatives are shown deterministically whereas the unclean attributes like Size, Engine and Condition with several possible clean values are shown as options. Only the first option in tuple t_3 matches the query. Thus the XOR result will contain only a single alternative for t_3 with probability 0.9. It is important to note that in the case of t_1, the Mid-size car can be associated with an Eng. value of I4 and a probability of 0.6 respectively. The XOR representation does not necessarily allow for combining Mid-size with either an Eng. value of V6 or a probability value of 0.4.

The experimental results compare the tuple ids when computing the recall of the method because tuple id provides the URL to the car's web page which can be used to determine a match. The output probabilistic relation is shown in Table III.

Table III. Result probabilistic database

TID	Model	Make	Orig.	Size	Eng.	Cond.	P
t_1	Civic	Honda	JPN	Mid-size/Compact	I4/I6	NEW	0.6/0.4
t_3	Civic	Honda	JPN	Mid-size	I4	USED	0.9

The interesting fact here is that the result of any query will always be a tuple-independent database. This is because we projected out every attribute except for the tuple-ID, and the tuple-IDs are independent of each other.

When showing the results of our experiments, we evaluate the precision and recall of the system. Since precision and recall are deterministic concepts, we have to convert the probabilistic database into a deterministic database (that will be shown to the user) prior to computing these values. We can do this conversion in two ways: (1) by picking only those tuples whose probability is higher than some threshold. We call this method the *threshold based determinization*. (2) by picking the top-k tuples and discarding the probability values (*top-k determinization*). The experiment section (Section 8.2) shows results with both determinizations.

6. QUERY REWRITING FOR ONLINE QUERY PROCESSING

In online query processing, we are primarily concerned with the scenario where the entire database is not available for cleaning: either due to access controls or due to size or frequency of change. In these cases, we only consider the problem of simple keyword queries over the data. More complex queries such as aggregates and joins require a comprehensive traversal of the dataset, which is by-definition hard in an online query scenario.

Algorithm 2 presents the online cleaning mode of BayesWipe. The first three operations comprising the sampling of the data, creation of the Bayes network from the sampled data, and the generation of error statistics are performed offline, and the remaining operations show how the tuples are efficiently retrieved from the database, ranked and displayed to the user. In this section we extend the techniques of the previous section so that it can be used in an online query processing method where the result tuples are cleaned at query time. Certain tuples that do not satisfy the query constraints, but are relevant to the user, need to be retrieved, ranked and shown to the user. The process also needs to be efficient, since the time that the users are willing to wait before results are shown to them is very small. We show our query rewriting mechanisms aimed at addressing both.

We begin by executing the user's query (Q^*) on the database. We store the retrieved results, but do not show them to the user immediately. We then find rewritten queries that are most likely to retrieve clean tuples. We do that in a two-stage process: we first

ALGORITHM 2: Algorithm for online query processing.

```
**Input**: $D$, the dirty dataset
**Input**: $Q$, the user's query
$S \leftarrow$ Sample the source dataset $D$
$BN \leftarrow$ Learn Bayes Network ($S$)
$ES \leftarrow$ Learn Error Statistics ($S$)
$R \leftarrow$ Query and score results ($Q, D, BN$)
$ESQ \leftarrow$ Get expanded queries ($Q$)
foreach Expanded query $E \in ESQ$ do
    $R \leftarrow R \cup$ Query and score results ($E, D, BN$)
    $RQ \leftarrow RQ \cup$ Get all relaxed queries ($E$)
end
$Sort(RQ)$ by expected relevance, using $ES$
while top-$k$ confidence not attained do
    $B \leftarrow$ Pick and remove top $RQ$
    $R \leftarrow R \cup$ Query and score results ($B, D, BN$)
end
$Sort(R)$ by score
return $R$
```

expand the query to increase the precision, and then relax the query by deleting some constraints (to increase the recall).

In the online query mode, we should take care to re-sample and relearn the model of the data occasionally, especially if it is suspected that the underlying model or characteristic of the data has changed. Such changes do occur in online data sources.

6.1. Increasing the precision of rewritten queries

We can improve precision by adding relevant constraints to the query Q^* given by the user. For example, when a user issues the query Model = Civic, we can expand the query to add relevant constraints Make = Honda, Country = Japan, Size = Mid-Size. These additions capture the essence of the query — because they limit the results to the specific kind of car the user is probably looking for. These expanded structured queries generated from the user's query are called ESQs. Querying using ESQs helps in eliminating $t4$ from the results in the motivating example from Table I in Section 1. This is because, it includes additional relevant constraints though the selection query constraint is only with respect to the attribute Model.

Each user query Q^* is a select query with one or more attribute-value pairs as constraints. In order to create an ESQ, we will have to add highly correlated constraints to Q^*.

Searching for correlated constraints to add requires Bayesian inference, which is an expensive operation. Therefore, when searching for constraints to add to Q^*, we restrict the search to the union of all the attributes in the Markov blanket [Pearl 1988]. The Markov blanket of an attribute comprises its children, its parents, and its children's other parents. It is the set of attributes whose value being given, the node becomes independent of all other nodes in the network. Thus, it makes sense to consider these nodes when finding correlated attributes. This correlation is computed using the Bayes Network that was learned offline on a sample database (recall the architecture of BayesWipe in Figure 1.)

Given a Q^*, we attempt to generate multiple ESQs that maximizes both the relevance of the results and the coverage of the queries of the solution space.

Note that if there are m attributes, each of which can take n values, then the total number of possible ESQs is n^m. Searching for the ESQ that globally maximizes the objectives in this space is infeasible; we therefore approximately search for it by performing a heuristic-informed search. Our objective is to create an ESQ with m attribute-value pairs as constraints.We begin with the constraints specified by the user query Q^*. We set these as evidence in the Bayes network, and then query the Markov blanket of these attributes for the attribute-value pairs with the highest posterior probability given this evidence. We take the top-k attribute-value pairs and append them to Q^* to produce k search nodes, each search node being a query fragment. If Q has p constraints in it, then the heuristic value of Q is given by $\mathbf{Pr}(Q)^{m/p}$. This represents the expected joint probability of Q when expanded to m attributes, assuming that all the constraints will have the same average posterior probability. We expand them further, until we find k queries of size m with the highest probabilities.

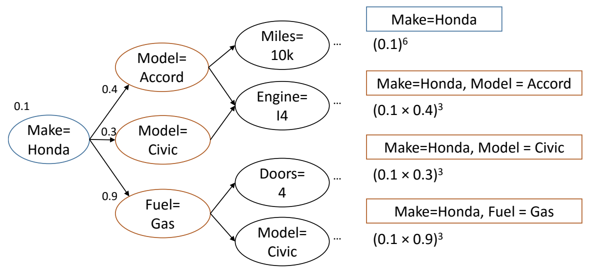


Fig. 3. Query Expansion Example. The tree shows the candidate constraints that can be added to a query, and the rectangles show the expanded queries with the computed probability values.

Example: In Figure 3, we show an example of the query expansion. The node on the left represents the query given by the user “Make=Honda”. First, we look at the Markov Blanket of the attribute Make, and determine that Model and Fuel are the nodes in the Markov blanket. we then set “Make=Honda” as evidence in the Bayes network and then run an inference over the values of the attribute Model. The two values of the Model attribute with the highest posterior probability are Accord and Civic. The most probable values of the Fuel attribute are “Gas” and “Electricity”. Using each of these values, new queries are constructed and added to the queue. Thus, the queue now consists of the 4 queries: “Make=Honda, Model=Civic”, “Make=Honda, Model=Accord” and “Make=Honda, Fuel=Gas”. A fragment of these queries are shown in the middle column of Figure 3. We dequeue the highest probability item from the queue and repeat the process of setting the evidence, finding the Markov Blanket, and running the inference. We stop when we get the required number of ESQs with a sufficient number of constraints.

6.2. Increasing the recall

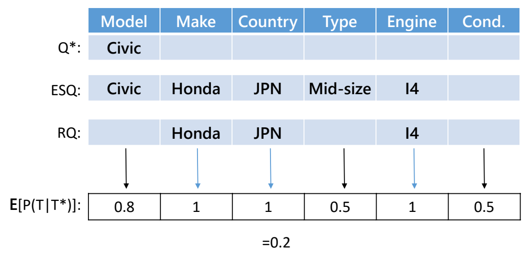
Adding constraints to the query causes the precision of the results to increase, but reduces the recall drastically. Therefore, in this stage, we choose to delete some constraints from the ESQs, thus generating the relaxed queries (RQ). Notice that the tuples that have corruptions in the attribute constrained by the user can only be retrieved by relaxed queries that do not specify a value for those attributes. Instead, we have to depend on rewritten queries that contain correlated values in other attributes to retrieve these tuples. Querying using RQs helps in including $t3$ and $t5$ in the results in the motivating example from Table I in Section 1. This is because, the selection constraint on the attribute Model is eliminated in the relaxed query and the correlated constraints on other attributes are considered to retain precision while enhancing the recall. Using relaxed queries can be seen as a trade-off between the recall of the result-set and the time taken, since there are an exponential number of relaxed queries for any given ESQ. As a result, an important question is the choice of RQs to execute. We take the approach of generating every possible RQ, and then ranking them according to their expected relevance. This operation is performed entirely on the learned error statistics, and is thus very fast.

We score each relaxed query by the *expected relevance* of its result set.

$$Rank(q) = \mathbb{E}\left(\frac{\sum_{T_q} \mathit{Score}(T_q|Q^*)}{|T_q|}\right)$$

where T_q are the tuples returned by a query q, and Q^* is the user's query. Executing an RQ with a higher rank will have a more beneficial impact on the result set because it will bring in better quality result tuples. Estimating this quantity is difficult because we do not have complete information about the tuples that will be returned for any query q. The best we can do, therefore, is to approximate this quantity.

Let the relaxed query be Q, and the expanded query that it was relaxed from be ESQ. We wish to estimate $\mathbb{E}[P(T|T^*)]$ where T are the tuples returned by Q. Using the attribute-error independence assumption, we can rewrite that as $\prod_{i=0}^{m} \mathbf{Pr}(T.A_i|T^*.A_i)$, where $T.A_i$ is the value of the i-th attribute in T. Since ESQ was obtained by expanding Q^* using the Bayes network, it has values that can be considered clean for this evaluation. Now, we divide the m attributes of the database into 3 classes: (1) The attribute is specified both in ESQ and in Q. In this case, we set $\mathbf{Pr}(T.A_i|T^*.A_i)$ to 1, since $T.A_i = T^*.A_i$. (2) The attribute is specified in ESQ but not in Q. In this case, we know what $T^*.A_i$ is, but not $T.A_i$. However, we can generate an average statistic of how often $T^*.A_i$ is erroneous by looking at our sample database. Therefore, in the offline learning stage, we precompute tables of error statistics for every T^* that appears in our sample database, and use that value. (3) The attribute is not specified in either ESQ or Q. In this case, we know neither the attribute value in T nor in T^*. We, therefore, use the average error rate of the entire attribute as the value for $\mathbf{Pr}(T.A_i|T^*.A_i)$. This statistic is also precomputed during the learning phase. This product gives the expected rank of the tuples returned by Q.



	Model	Make	Country	Type	Engine	Cond.
Q*:	Civic					
ESQ:	Civic	Honda	JPN	Mid-size	I4	
RQ:		Honda	JPN		I4	
E[P(T|T*)]:	0.8	1	1	0.5	1	0.5

=0.2

Fig. 4. Query Relaxation Example.

Example: In Figure 4, we show an example for finding the probability values of a relaxed query. Assume that the user's query Q^* is "Civic", and the ESQ is shown in the second row. For an RQ that removes the attribute values "Civic" and "Mid-Size" from the ESQ, the probabilities are calculated as follows: For the attributes "Make, Country" and "Engine", the values are present in both the ESQ as well as the RQ, and therefore, the $P(T|T^*)$ for them is 1. For the attribute "Model" and "Type", the values are present in ESQ but not in RQ, hence the value for them can be computed from the learned error statistics. For example, for "Civic", the average value of $P(T|Civic)$ as learned from the sample database (0.8) is used. Finally, for the attribute "Condition", which is present neither in ESQ nor in RQ, we use the average error statistic for that attribute (i.e. the average of $P(T_a|T_a^*)$ for a = "Condition" which is 0.5).

The final value of $\mathbb{E}[P(T|T^*)]$ is found from the product of all these attributes as 0.2. This process is very fast because it only involves lookups and multiplication – bayesian inference is not needed.

6.3. Terminating the process

We begin by looking at all the RQs in descending order of their rank. If the current k-th tuple in our resultset has a relevance of λ, and the estimated rank of the Q we are about to execute is $R(T_q|Q)$, then we stop evaluating any more queries if the probability $\mathbf{Pr}(R(T_q|Q) > \lambda)$ is less than some user defined threshold $\mathcal{P}$. This ensures that we have the true top-k resultset with a probability $\mathcal{P}$.

7. MAP-REDUCE FRAMEWORK

BayesWipe is most useful for big-data related scenarios. BayesWipe has two modes: online and offline. The online mode of BayesWipe already works for big data scenarios by optimising the rewritten queries it issues. Now, we show that the offline mode can also be optimized for a big-data scenario by implementing it as a Map-Reduce application. In this section, we aim to show that BayesWipe can indeed be parallelized using a Map-Reduce framework and explain how it can be done. Our aim is not necessarily to show the optimization for efficiency and hence, the associated experiments were run on sample datasets using a natively implemented Map-Reduce framework in C# and not on out-of-the-box products like Hadoop. The implementation was tested on multiple processes spawned on a single machine.

So far, BayesWipe-Offline has been implemented as a two-phase, single threaded program. In the first phase, the program learns the Bayes network (both structure and parameters), learns the error statistics, and creates the candidate index. Recall from section 4.3 that we create an index on the attributes of the sample database to speed up the creation of the candidate set of clean tuples; which we refer to as the candidate index. The candidate index is constructed on a set of $j+1$ attributes when the restriction on a candidate clean tuple is to differ from the dirty tuple in not more than j attributes. The attributes in the dirty tuple are compared to the attributes of the tuples in the sample database using the candidate index to generate the set of candidate clean tuples. Note that this candidate index can be constructed on any arbitrary set of $j+1$ attributes present in the sample database. In the second phase, the program goes through every tuple in the input database, picks a set of candidate tuples, and then evaluates the $P(T^*|T)P(T^*)$ for every candidate tuple, and replaces T with the T^* that maximises that value. Since the learning is typically done on a sample of the data, it is more important to focus on the second phase for the parallelizing efforts. Later, we will see how the learning of the error statistics can also be parallelized.

7.1. Simple Approach

The simplest approach to parallelizing BayesWipe is to run the first phase (the learning phase) on a single machine. Then, a copy of the bayes network (structure and CPTs), the error statistics, and the candidate index can be sent to a number of other machines. Each of those machines also receives a fraction of the input data from the dirty database. With the help of the generative model and the input data, it can clean the tuples, and then create the output.

If we express this in Map-Reduce terminology, we will have a pre-processing step where we create the generative and error models. The Map-Reduce architecture will have only mappers, and no reducers. The result of the mapping will be the tuple $\langle T, T^* \rangle$.

The problem with this approach is that in a truly big data scenario, the candidate index can become very large. Indeed, as the number of tuples increases, the size of the domain of each attribute also increases (see Figure 8(a) for 1 shard). Further, the number of different combinations, and the number of erroneous values for each attribute also increase (Figure 8(b)). All of this results in a rather large candidate index. Transmitting and using the entire index on each mapper node is wasteful of both network, memory, (and if swapped out, disk resources). Note that to create a rich and useful data correction system, we have to accommodate a large candidate clean-tuple set, $\mathcal{C}(T)$, for every T. $\mathcal{C}(T)$ roughly tracks the sample database size. If we are unable to shard $\mathcal{C}(T)$, then sharding the input data is pointless. In the following section we endeavor to show a strategy where not just the input, but also the index on the candidate set $\mathcal{C}(T)$ can be sharded across machines.

7.2. Improved Approach

In order to split both the input tuples and the candidate index, we use a two-stage approach. In the first stage, we run a map-reduce that splits the problem into multiple shards, each shard having a small fraction of the candidate index. The second stage is a simple map-reduce that picks the best output from stage 1 for each input tuple.

Stage 1: Given an input tuple T and a set of candidate tuples, the T^*s, suppose the candidate index is created on k attributes, $A_1 ... A_k$. We can say that for every tuple T, and one of its candidate tuples T^*, they will have at least one matching attribute a_i from this set. We can use this common element a_i to predict which shards the candidate T^*s might be available in. We therefore, send the tuple T to each shard that matches the hash of the value a_i.

In the map-reduce architecture, it is possible to define a 'partition' function. Given a mapped key-value pair, this function determines which reducer nodes will process the data. We can use an exact equivalence on each value that the matching attribute can take, a_i as the partition function. However, notice that the number of possible values that $A_1 ... A_k$ can take is rather large. If we naïvely use a_i as the partition function, we will have to create those many reducer nodes. Therefore, more generally, we hash this value into a fixed number of reducer nodes, using a deterministic hash function. This will then find all candidate tuples that are eligible for this tuple, compute the similarity, and output it.

Example: Suppose we have tuple T_1 that has values $(a_1, a_2, a_3, a_4, a_5)$. Suppose our candidate index is created on attributes A_1, A_2, A_4. This means that any candidates T^* that are eligible for this tuple have to match one of the values a_1, a_2 or a_4. Then the mapper will create the pairs (a_1, T), (a_2, T) and (a_4, T), and send to the reducers. The partition function is the hash of the key – so in this case, the first one will be sent to the reducer number $hash(A1 = a1)$, the second will be sent to the reducer numbered $hash(A2 = a2)$, and so on.

In the reducer, the similarity computation and computation of the prior probabilities from the BayesWipe algorithm are run. Since each reducer only has a fraction of the candidate index (the part that matches $A1 = a1$, for instance), it can hold it in memory and computation is quite fast. Each reducer produces a pair $(T_1, (T^*_{1n}, \text{score}))$. Since there are several candidate clean tuples, n is used to identify a specific tuple among those alternatives.

Stage 2: This stage is a simple $\max$ calculation. The mapper does nothing, it simply passes on the key-value pair $(T_1, (T^*_{1n}, \text{score}))$ that was generated in the previous Map-Reduce job. Notice that the key of this pair is the original, dirty tuple T_1. The Map-Reduce architecture thus automatically groups together all the possible clean versions of T_1 along with their scores. The reducer picks the best T* based on the score (using a simple $\max$ function), and outputs it to the database.

7.3. Results of This Strategy

In Figure 8(a) and Figure 8(b) we can see how this map reduce strategy helps in reducing the memory footprint of the reducer. First, we plot the size of the index that needs to be held in each node as the number of tuples in the input increases. The top-

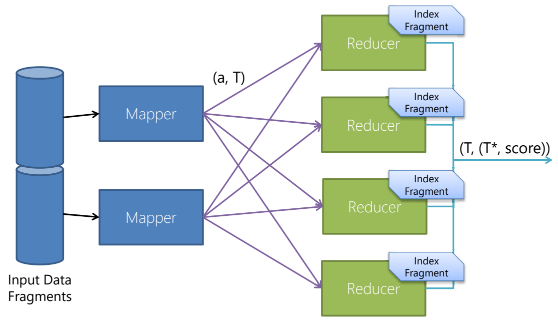


Fig. 5. Stage-1 Map-Reduce Framework for BayesWipe.

most curve shows the size of index in bytes if there was no sharding – as expected, it increases sharply. The other curves show how the size of the index in the one of the nodes varies for the same dataset sizes. From the graph, it can be seen that as the number of tuples increases, the size of the index grows at a lower rate when the number of shards is increased. This shows that increasing the number of reduce nodes is a credible strategy for distributing the burden of the index.

In the second figure (Figure 8(b)), we see how the size of the index varies with the percentage of noise in the dataset. As expected, when the noise increases, the number of possible candidate tuples increase (since there are more variations of each attribute value in the pool). Without sharding, we see that the size of the dataset increases. While the increase in the size of the index is not as sharp as the increase due to the size of the dataset, it is still significant. Once again, we observe that as the number of shards is increased, the size of the index in the shard reduces to a much more manageable value.

Note that a slight downside to this architecture is that a shuffle/reduce of $(T, (T^*_n, \text{score}))$ is needed in the second stage, and this intermediate data structure can be quite large. While this leads to some network and temporary storage overhead, the primary objective of sharding the expensive computation has been achieved by this architecture.

8. EMPIRICAL EVALUATION

We quantitatively study the performance of BayesWipe in both its modes — offline, and online, and compare it against state-of-the-art CFD approaches.

We present experiments on evaluating the approach in terms of the effectiveness of data cleaning, efficiency and precision of query rewriting. All the experiments were run on a Dell Optiplex machine with a 64-bit Windows 7 Operating System and 4GB of RAM. The software used was C#, Microsoft .NET framework 3.5, Visual Studio 2013, Banjo [Hartemink. 2005] and Infer.NET [Minka et al. 2010]. A demo for the offline cleaning mode of BayesWipe can be downloaded from http://bayeswipe.sushovan.de/.

8.1. Datasets

To perform the experiments, we obtained the real data from the web. The first dataset is *Used car sales* dataset D_{car} crawled from Google Base of approximately 30k tuples with 8 attributes. Such “dirty” dataset is referred to as “D'_{car}”. The second dataset we used was adapted from the *Census Income* dataset D_{census} from the UCI machine learning repository [Asuncion and Newman 2007]. From the fourteen available attributes, we picked the attributes that were categorical in nature, resulting in the following 8 attributes: working-class, education, marital status, occupation, race, gender, filing status. country. The same setup was used for both datasets – including parameters and error features.

These datasets were observed to be mostly clean. We then introduced[5] three types of noise to the attributes. To add noise to an attribute, we randomly changed it either to a new value which is close in terms of string edit distance (distance between 1 and 4, simulating spelling errors) or to a new value which was from the same attribute (simulating replacement errors) or just delete it (simulating deletion errors). As we have mentioned before, one of the assumptions of this paper is that the error model is a combination of these three kinds of errors, and that the errors are independent of each other. By synthetically generating these errors, we were able to test our system against a dataset that satisfies the assumption.

The next dataset tests our system against a real-world scenario where we do not control the error process, and thus validates that our error model is not unrealistic.

To test our system against real-world noise where we do not have any control over amount, type or behavior of the noise generation process, we crawled car inventory data from the website 'cars.com', obtaining 1.2 million tuples with 15 attributes. We manually verified that the data obtained did, in fact, have a reasonable number of inaccuracies, making it a suitable candidate for testing our system.

Broad applicability to various datasets: We rely on a Bayes Network to learn the dependencies in the data, which means we inherit the strengths and weaknesses of the structure and parameter learning tools. Both the packages we used for learning the Bayes Network: Banjo and Infer.NET learned reasonable structures for the various datasets we used using default settings. Notice that the Bayes Network learned from these datasets was shown earlier in Figure 2. Intuitively we see that learned structure is correct.

Additionally, the only change we made to the datasets obtained directly from the source was to introduce noise.

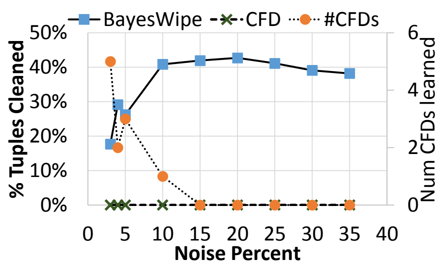
8.2. Experiments

Offline Cleaning Evaluation: The first set of evaluations shows the effectiveness of the offline cleaning mode on a single relational table containing 20,000 tuples in total out of which there are close to 8,000 dirty tuples. In Figure 6(a), we compare BayesWipe against CFDs [Chiang and Miller 2008].

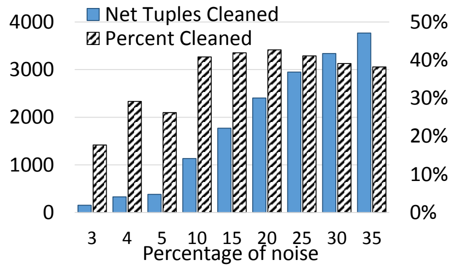
The dotted line that shows the number of CFDs learned from the noisy data quickly falls to zero, which is not surprising: CFDs learning was designed with a clean training dataset in mind. Even if there is a single dirty tuple in the data sample violating the pattern learnt, the CFD cannot hold. Further, the only constraints learned by this algorithm from the dirty sample are the ones that have not been violated in the entire dataset — unless a tuple violates some CFD, it cannot be cleaned. As a result, the CFD method cleans exactly zero tuples independent of the noise percentage. On the other hand, BayesWipe is able to clean between 20% to 40% of the incorrect values. This is because it not only learns attribute correlations from the majority of the data sample, but it also implicitly weights the data source tuples with their prior probabilities. This is combined with the error likelihood (their distance from the observed dirty tuples). It is interesting to note that the percentage of tuples cleaned increases initially and then slowly decreases, because for very low values of noise, there are not enough errors for the system to learn a reliable error model from; and at larger values of noise, the data source model learned from the noisy data is of poorer quality.

While Figure 6(a) showed only percentages, in Figure 6(b) we report the actual number of tuples cleaned in the dataset along with the percentage cleaned. This curve

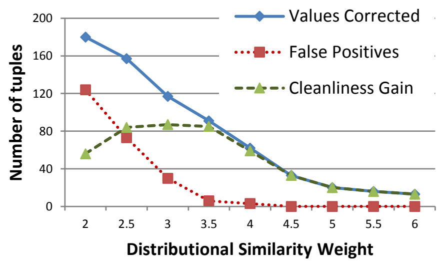
[5]We note that the introduction of synthetic errors into clean data for experimental evaluation purposes is common practice in data cleaning research [Cong et al. 2007; Bohannon et al. 2007].



(a) % performance of BayesWipe compared to CFD, for the used-car dataset.



(b) % net corrupt values cleaned, car database



(c) Net corrections vs γ

Fig. 6. Offline cleaning mode of BayesWipe

shows that the raw number of tuples cleaned always increases with higher input noise percentages.

Setting γ: As explained in Section 5.1, the weight given to the edit distance (δ) compared to the weight given to the distributional similarity ($1-\delta$); and the overcorrection parameter (γ) are parameters that can be tuned, and should be set based on which kind of error is more likely to occur. In our experiments, we performed a grid search to determine the best values of δ and γ to use. In Figure 6(c), we show a portion of the grid search by varying γ and keeping $\delta = 2/5$ fixed.

The "values corrected" data points in the graph correspond to the number of erroneous attribute values that the algorithm successfully corrected (when checked against the ground truth). The "false positives" are the number of legitimate values that the algorithm changes to an erroneous value. When cleaning the data, our algorithm chooses a candidate tuple based on both the prior of the candidate as well as the likelihood of the correction given the evidence. Low values of γ give a higher weight to the prior than the likelihood, allowing tuples to be changed more easily to candidates with high prior. The "overall gain" in the number of clean values is calculated as the difference of clean values between the output and input of the algorithm.

Given ground truth in the form of training data, this grid search can be done very effectively, but under the absence of ground truth, BayesWipe can still be used to a good approximation using simply an estimate of the noise percentage in the data. Instead of maximizing the cleanliness gain using a grid search, we attempt to match the total number of tuples modified by the algorithm as closely as possible to this estimated noise percentage. It yields a good first approximation for the value of the parameters. This can be further tweaked later when more information is available about the result quality.

If we set the parameter values too low, we will correct most wrong tuples in the input dataset, but we will also 'overcorrect' a larger number of tuples. If the parameters are set too high, then the system will not correct many errors — but the number of 'overcorrections' will also be lower. Based on these experiments, we picked a parameter value of $\delta = 0.638, \gamma = 5.8$ and kept it constant throughout.

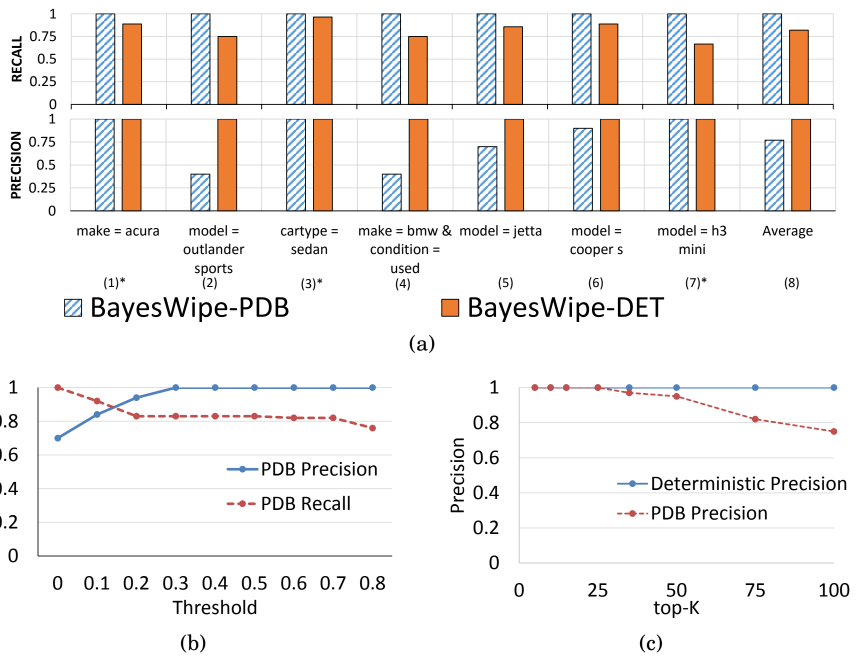


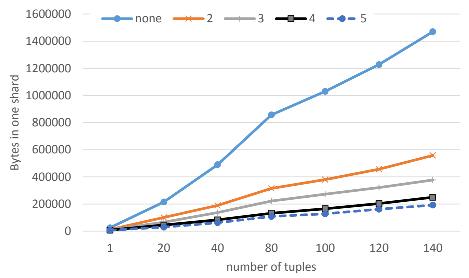
Fig. 7. Results of probabilistic method.

Using probabilistic databases: We empirically evaluate the PDB-mode of BayesWipe in Figure 7. In the first figure, we show the performance of the PDB mode of BayesWipe, BayesWipePDB, against the deterministic mode, BayesWipeDET, for specific queries. As we can see from the first, third and seventh queries (marked with an asterisk in the figure), the BayesWipe-PDB improves the recall without any loss of precision. However, in most cases (and on average), BayesWipe-PDB provides a better recall at the cost of some precision.

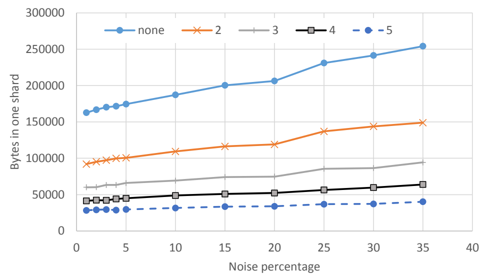
The second figure shows the performance of BayesWipe-PDB as the probability threshold for inclusion of a tuple in the resultset is varied. As expected, with low values of the threshold, the system allows most tuples into the resultset, thus showing high recall and low precision. As the threshold increased, the precision increases, but the recall falls.

In Figure 7(c), we compare the precision of the PDB mode using top-k determinization against the deterministic mode of BayesWipe. As expected, both the modes show high precision for low values of k, indicating that the initial results are clean and relevant to the user. For higher values of k, the PDB precision falls off, indicating that PDB methods are more useful for scenarios where high recall is important without sacrificing too much precision.

Online Query Processing: While in the offline mode, we had the luxury of changing the tuples in the database itself, in online query processing, we use query rewriting to obtain a resultset that is similar to the offline results, without modification to the database. We consider an SQL select query system as our baseline (see Section 3 for an example). We evaluate the precision and recall of our method against the ground truth and compare it with the baseline, using randomly generated queries.

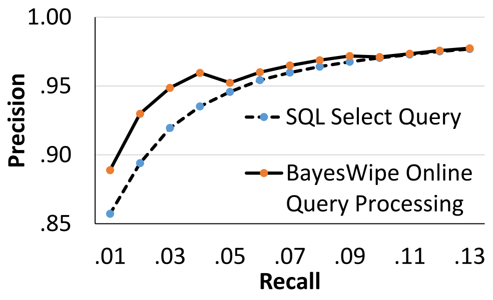


(a) Bytes in one shard vs the Number of Tuples (in Thousands), for Various Numbers of Shards.

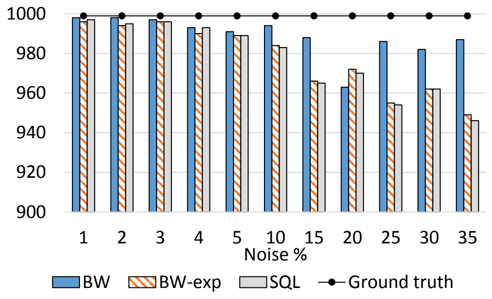


(b) Bytes in one shard vs the Noise percentage, for Various Numbers of Shards.

Fig. 8. Map-Reduce index sizes



(a) Average precision vs recall with 20% noise



(b) Net improvement in data quality (TP-FP)

Fig. 9. Online cleaning mode of BayesWipe

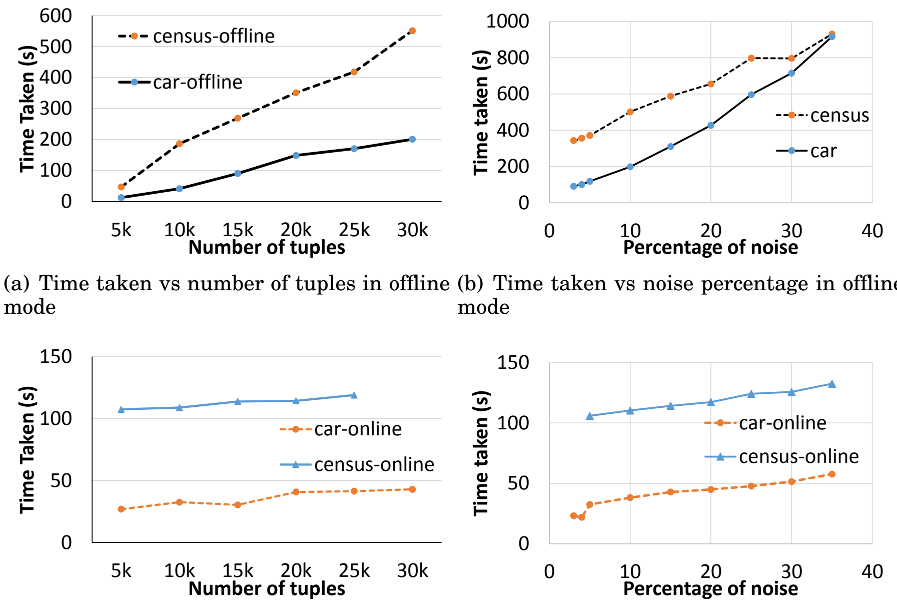
We issued randomly generated queries to both BayesWipe and the baseline system. Figure 9(a) shows the average precision over 10 queries at various recall values. It shows that our system outperforms the SQL select query system in top-k precision, especially since our system considers the relevance of the results when ranking them. On the other hand, the SQL search approach is oblivious to ranking and returns all tuples that satisfy the user query. Thus it may return irrelevant tuples early on, leading to less precision.

Figure 9(b) shows the improvement in the absolute numbers of tuples returned by the BayesWipe system. The graph shows the number of true positive tuples returned (tuples that match the query results from the ground truth) minus the number of false positives (tuples that are returned but do not appear in the ground truth result set). We also plot the number of true positive results from the ground truth, which is the theoretical maximum that any algorithm can achieve. The graph shows that the BayesWipe system outperforms the SQL query system at nearly every level of noise. Further, the graph also illustrates that — compared to an SQL query baseline — BayesWipe closes the gap to the maximum possible number of tuples to a large extent. In addition to showing the performance of BayesWipe against the SQL query baseline, we also show the performance of BayesWipe without the query relaxation part (called BW-exp[6]). We can see that the full BayesWipe system outperforms the BW-exp system significantly, showing that query relaxation plays an important role in bringing relevant tuples to the resultset, especially for higher values of noise.

This shows that our proposed query ranking strategy indeed captures the expected relevance of the to-be-retrieved tuples, and the query rewriting module is able to generate the highly ranked queries.

Efficiency: In Figure 10 we show the data cleaning time taken by the system in its various modes. The first two graphs show the offline mode, and the second two show the online mode. As can be seen from the graphs, BayesWipe performs reasonably well both in the datasets of large size and the datasets with large noise.

The time required to clean the database increases with increased noise percentage of the tuples. When the percentage of noise increases, we are more likely to see new unique values in each attribute. This is because naturally occurring noise is random, resulting in values that don't occur elsewhere in the database. This increases the space over which the algorithms have to search. In the offline algorithm, the increase is contributed to by both the learning phase and the cleaning phase; in the learning phase there is a larger space of possible values to learn from, and in the cleaning phase there is a much larger candidate set of values to evaluate. For the online algorithm, the increase is present, but less pronounced because new values observed during the online cleaning phase are not considered as candidates for other input tuples.



(a) Time taken vs number of tuples in offline mode

(b) Time taken vs noise percentage in offline mode

(c) Time taken vs number of tuples in online mode

(d) Time taken vs noise percentage in online mode

Fig. 10. Performance evaluations

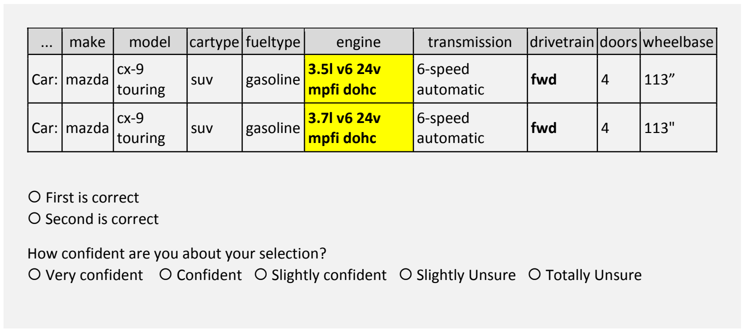
Evaluation on real data with naturally occurring errors: In this section we used a dataset of 1.2 Million tuples crawled from the cars.com website[7] to check the perfor-

[6]BW-exp stands for BayesWipe-expanded, since the only query rewriting operation done is query expansion.
[7]http://www.cars.com

mance of the system with real-world data, where the corruptions were not synthetically introduced. Since this data is large, and the noise is completely naturally occurring, we do not have ground truth for this data. To evaluate this system, we conducted an experiment on Amazon Mechanical Turk. First, we ran the offline mode of BayesWipe on the entire database. We then picked only those tuples that were changed during the cleaning, and then created an interface in mechanical turk where only those tuples were shown to the user in random order. Due to resource constraints, the experiment was run with the first 200 tuples that the system found to be unclean. We inserted 3 known answers into the questionnaire, and removed any responses that failed to annotate at least 2 out of the 3 answers correctly.

Table IV. Results of the Mechanical Turk Experiment, showing the percentage of tuples for which the users picked the results obtained by BayesWipe as against the original tuple. Also shows performance against a random modification.

Confidence	BayesWipe	**Original**	**Random**	**Increase over Random**
High confidence only	56.3%	43.6%	5.5%	50.8% points (10x better)
All confidence values	53.3%	46.7%	12.4%	40.9% points (4x better)



...	make	model	cartype	fueltype	engine	transmission	drivetrain	doors	wheelbase
Car:	mazda	cx-9 touring	suv	gasoline	**3.5l v6 24v mpfi dohc**	6-speed automatic	**fwd**	4	113”
Car:	mazda	cx-9 touring	suv	gasoline	**3.7l v6 24v mpfi dohc**	6-speed automatic	**fwd**	4	113"

○ First is correct
○ Second is correct

How confident are you about your selection?
○ Very confident ○ Confident ○ Slightly confident ○ Slightly Unsure ○ Totally Unsure

Fig. 11. A fragment of the questionnaire provided to the Mechanical Turk workers.

An example is shown in Figure 11. The turker is presented with two cars, and she does not know which of the cars was originally present in the dirty dataset, and which one was produced by BayesWipe. The turker will use her own domain knowledge, or perform a web search and discover that a Mazda CX-9 touring is only available in a 3.7l engine, not a 3.5l. Then the turker will be able to declare the second tuple as the correct option with high confidence.

The results of this experiment are shown in Table IV. As we can see, the users consistently picked the tuples cleaned by BayesWipe more favorably compared to the original dirty tuples, proving that it is indeed effective in real-world datasets. Notice that it is not trivial to obtain a 56% rate of success in these experiments. Finding a tuple which convinces the turkers that it is better than the original requires searching through a huge space of possible corrections. An algorithm that picks a possible correction randomly from this space is likely to get a near 0% accuracy.

The first row of Table IV shows the fraction of tuples for which the turkers picked the version cleaned by BayesWipe and indicated that they were either ‘very confident’ or ‘confident’. The second row shows the fraction of tuples for all turker confidence values, and therefore is a less reliable indicator of success.

In order to show the efficacy of BayesWipe we also performed an experiment in which the same tuples (the ones that BayesWipe had changed) were modified by a random perturbation. The random perturbation was done by the same error process as described before (typo, deletion, substitution with equal probability). Then these tuples (the original tuple from the database and the perturbed tuple) were presented as two choices to the turkers. The preference by the turkers for the randomly perturbed tuple over the original dirty tuple is shown in the third column, 'Random'. It is obvious from this that the turkers overwhelmingly do not favor the random perturbed tuples. This demonstrates two things. First, it shows the fact that BayesWipe was performing useful cleaning of the tuples. In fact, BayesWipe shows a tenfold improvement over the random perturbation model, as judged by human turkers. This shows that in the large space of possible modifications of a wrong tuple, BayesWipe picks the correct one most of the time. Second, it provides additional support for the fact that the turkers are picking the tuple carefully, and are not randomly submitting their responses.

In this experiment, we also found the average fraction of known answers that the turkers gave wrong answers to. This value was 8%. This leads to the conclusion that the difference between the turker's preference of BayesWipe over both the original tuples (which is 12%) and the random perturbation (which is 50%) are both significant.

9. CONCLUSION

In this paper we presented a novel system, BayesWipe that works using end-to-end probabilistic semantics, and without access to clean master data. We showed how to effectively learn the data source model as a Bayes network, and how to model the error as a mixture of error features. We showed the operation of this system in two modalities: (1) offline data cleaning, an *in situ* rectification of data and (2) online query processing mode, a highly efficient way to obtain clean query results over inconsistent data. There is an option to generate a standard, deterministic database as the output, as well as a probabilistic database, where all the alternatives are preserved for further processing. We empirically showed that BayesWipe outperformed existing baseline techniques in quality of results, and was highly efficient. We also showed the performance of the BayesWipe system at various stages of the query rewriting operation. We demonstrated how BayesWipe can be run on the map-reduce architecture so that it can scale to huge data sizes. User experiments showed that the system is useful in cleaning real-world noisy data.

REFERENCES

Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *PODS*. ACM, 68–79.

A. Asuncion and D.J. Newman. 2007. UCI machine learning repository. (2007).

A.L. Berger, V.J.D. Pietra, and S.A.D. Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* (1996).

Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2011. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*.

George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2013a. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*. IEEE.

George Beskales, Ihab F Ilyas, Lukasz Golab, and Artur Galiullin. 2013b. Sampling from repairs of conditional functional dependency violations. *The VLDB Journal* (2013), 1–26.

P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*. ACM.

P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *ICDE*. IEEE, 746–755.

Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. 2005. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*. 891–893.

Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.

F. Chiang and R.J. Miller. 2008. Discovering data quality rules. *Proceedings of the VLDB Endowment* (2008).

Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 1247–1261. DOI:http://dx.doi.org/10.1145/2723372.2749431

Computing Research Association. 2012. Challenges and Opportunities with Big Data. http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf. (2012).

G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. 2007. Improving data quality: Consistency and accuracy. In *VLDB*. VLDB Endowment, 315–326.

G. Cormode, L. Golab, K. Flip, A. McGregor, D. Srivastava, and X. Zhang. 2009. Estimating the confidence of conditional functional dependencies. In *Proceedings of the 35th SIGMOD international conference on Management of data*. ACM, 469–482.

Nilesh Dalvi and Dan Suciu. 2004. Efficient query evaluation on probabilistic databases. In *VLDB*, Vol. 30. VLDB Endowment, 864–875.

Tamraparni Dasu and Ji Meng Loh. 2012. Statistical distortion: Consequences of data cleaning. *VLDB* 5, 11 (2012), 1674–1683.

Sushovan De. 2014. *Unsupervised Bayesian Data Cleaning Techniques For Structured Data*. Ph.D. Dissertation. Arizona State University.

Sushovan De, Yuheng Hu, Yi Chen, and Subbarao Kambhampati. 2014. BayesWipe: A multimodal system for data cleaning and consistent query answering on structured bigdata. In *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 15–24.

Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. 2009. Truth discovery and copying detection in a dynamic world. *VLDB* 2, 1 (2009), 562–573.

Wenfei Fan and Floris Geerts. 2012. Foundations of Data Quality Management. *Synthesis Lectures on Data Management* 4, 5 (2012), 1–217.

Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2 (2008), 6.

W. Fan, F. Geerts, L. Lakshmanan, and M. Xiong. 2009. Discovering conditional functional dependencies. In *ICDE*. IEEE.

W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. 2010. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment* (2010).

I.P. Fellegi and D. Holt. 1976. A systematic approach to automatic edit and imputation. *J. American Statistical association* (1976), 17–35.

Ariel Fuxman, Elham Fazli, and Renée J Miller. 2005. Conquer: Efficient management of inconsistent databases. In *SIGMOD*. ACM, 155–166.

Lukasz Golab, Howard J. Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.

Patrick Gray. 2013. Before Big Data, clean data. (2013). http://www.techrepublic.com/blog/big-data-analytics/before-big-data-clean-data/

A. Hartemink. 2005. Banjo: Bayesian network inference with java objects. http://www.cs.duke.edu/~amink/software/banjo. (2005).

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke S. Zettlemoyer, and Daniel S. Weld. 2011. Knowledge-Based Weak Supervision for Information Extraction of Overlapping Relations. In *ACL*. The Association for Computer Linguistics, 541–550.

Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. 2010. Learning 5000 Relational Extractors. In *ACL*. The Association for Computer Linguistics, 286–295.

Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. 2011. Research Directions in Data Wrangling: Visuatizations and Transformations for Usable and Credible Data. *Information Visualization* (2011), 271–288.

E.M. Knorr, R.T. Ng, and V. Tucakov. 2000. Distance-based outliers: algorithms and applications. *The VLDB Journal* 8, 3 (2000), 237–253.

Jeremy Kubica and Andrew Moore. 2003. Probabilistic Noise Identification and Data Cleaning. In *ICDM*. IEEE, 131–138.

Heather Leslie. 2010. Health data quality – a two-edged sword. (2010). http://omowizard.wordpress.com/2010/02/21/health-data-quality-a-two-edged-sword/

M. Li, Y. Zhang, M. Zhu, and M. Zhou. 2006. Exploring distributional similarity based models for query spelling correction. In *ICCL*. Association for Computational Linguistics, 1025–1032.

Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2009. A Statistical Method for Integrated Data Cleaning and Imputation. *Purdue University Computer Science Technical Reports* (2009).

T. Minka, Win J.M., J.P. Guiver, and D.A. Knowles. 2010. Infer.NET 2.4. (2010). Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.

Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB*. Morgan Kaufmann Publishers Inc., 381–390.

E.S. Ristad and P.N. Yianilos. 1998. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (1998).

S.J. Russell and P. Norvig. 2010. *Artificial intelligence: a modern approach*. Prentice Hall.

Parag Singla and Pedro Domingos. 2006. Entity resolution with markov logic. In *ICDM*. IEEE, 572–582.

Dan Suciu and Nilesh Dalvi. 2005. Foundations of probabilistic answers to queries. *SIGMOD* 14, 16 (2005), 963–963.

Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1483–1494.

Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. 2014. A Sample-and-clean Framework for Fast and Accurate Query Processing on Dirty Data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 469–480. DOI:http://dx.doi.org/10.1145/2588555.2610505

Jiannan Wang and Nan Tang. 2014. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 457–468.

Tim Weninger and Jiawei Han. 2013. Exploring structure and content on the web: extraction and integration of the semi-structured web. In *WSDM*. ACM, 779–780.

Garrett Wolf, Aravind Kalavagattu, Hemal Khatri, Raju Balakrishnan, Bhaumik Chokshi, Jianchun Fan, Yi Chen, and Subbarao Kambhampati. 2009. Query processing over incomplete autonomous databases: query rewriting using learned data dependencies. *The VLDB Journal* (2009).

Hui Xiong, Gaurav Pandey, Michael Steinbach, and Vipin Kumar. 2006. Enhancing data analysis with noise removal. *TKDE* 18, 3 (2006), 304–319.

Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. 2011. Guided data repair. *VLDB* 4, 5 (2011), 279–289.

Ce Zhang. 2015. *DeepDive: A data management system for automatic knowledge base construction*. Ph.D. Dissertation. University of Wisconsin–Madison.

Congle Zhang, Raphael Hoffmann, and Daniel S. Weld. 2012. Ontological Smoothing for Relation Extraction with Minimal Supervision. In *AAAI*. AAAI Press.

Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. 2015. QASCA: A Quality-Aware Task Assignment System for Crowdsourcing Applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1031–1046.