Using Markov Chain Monte Carlo to Play Trivia

Daniel Deutch [1] Ohad Greenshpan [1,2] Boris Kostenko [1] Tova Milo [1]

[1] *Tel-Aviv University* [2] *IBM Haifa Research Labs*

***Abstract*—We introduce in this Demonstration a system called *Trivia Masster* that generates a very large Database of facts in a variety of topics, and uses it for question answering. The facts are collected from human users (the "crowd"); the system motivates users to contribute to the Database by using a Trivia *Game*, where users gain points based on their contribution. A key challenge here is to provide a suitable Data Cleaning mechanism that allows to identify which of the facts (answers to Trivia questions) submitted by users are indeed correct / reliable, and consequently how many points to grant users, how to answer questions based on the collected data, and which questions to present to the Trivia players, in order to improve the data quality. As no existing single Data Cleaning technique provides a satisfactory solution to this challenge, we propose here a novel approach, based on a declarative framework for defining recursive and probabilistic Data Cleaning rules. Our solution employs an algorithm that is based on Markov Chain Monte Carlo Algorithms.**

I. Introduction

Harnessing a *crowd* of users for the collection of mass data and for solving problems has recently become a wide-spread technique [1]. Specifically, the work of [2], [3] suggested the use of *games* as a tool that attracts crowd to collaboratively perform a wide range of difficult tasks. In the internet era, such techniques have the potential of generating large Databases of facts, that is otherwise very difficult to construct. To fulfill this potential, though, one must overcome two main challenges. The first is *motivating* the crowd to contribute to the Database in a helpful manner. The second stems from the fact that data supplied by the crowd may be erroneous or contradicting. This calls for *Data Cleaning*. A key difficulty lies in the *recursive* dependencies between these two challenges: to motivate users to contribute in a helpful manner we must first know which data pieces are correct, which require validation, and which are completely missing. Thus we should perform Data Cleaning. But in order to know how to clean the data, we should know which users are trustable, and this depends on how much of their submitted data was correct. This calls for *recursive* definitions of cleaning rules. Furthermore, there are many possible cleaning options for every given Database, and we cannot be certain neither which option is the "most correct", nor which users are more trustable for a given fact. This calls for *probabilistic* choices out of the various options.

We demonstrate *Trivia Masster*, whose goal is to generate a very large Database of facts in a variety of topics. The facts are collected from a crowd of users, and are then used to answer queries. At a high-level, the system consists of two main components. The first component is a Trivia Game, called *GUESS What*, which is designed to attract users to supply data in various areas. The game is run in rounds, in each of which users are given a question (e.g. names of world capital cities; or "What is the capital city of China?") and are required to provide one or more answers. The answers given by the users are stored in the Database. To further motivate users to contribute *correct and new* pieces of information, we use a *scoring* mechanism, assigning high scores to users that contribute useful facts, as well as a mechanism that decides *which questions to ask*. The second component is a Query Answering mechanism. It receives as input a (form-based) query over the collected Data, and decides what is the correct answer, based on *data cleaning* rules.

Many approaches for Data Cleaning have been proposed in the literature. For example, a simple approach decides between two contradicting facts according to their support [4]; another approach suggests the application of "transformation" rules [5] that fix parts of the data. A recent paper [6] suggests to gradually clean data based on "corroboration", i.e. the trust the system has in the users providing it. Unfortunately, the above mentioned techniques are hard-coded and are thus inflexible and difficult to combine and optimize; in particular, no single technique is guaranteed to always achieve superior results. What is desirable here is to have a declarative framework for describing such cleaning strategies, which allows for rapid adjustments, modification and optimization. The current declarative frameworks (e.g. [7]) either support only probabilistic rules, or only recursion, but *not both*.

The design of a novel declarative framework for recursive, probabilistic data cleaning rules thus stands at the core of our contribution. This framework is demonstrated in the context of *Trivia Masster*, but it is important to note that the framework is generic and may be applied in any context where such recursive and probabilistic rules are suitable. Our framework uses data cleaning rules defined in relational algebra, augmented by a particular operator that allows to introduce probabilities, and supports recursive rule invocation. We showed in [8] that evaluation of these rules is possible via an adaptation of *Markov Chain Monte Carlo algorithms* [9], where, intuitively the Markov Chain states correspond to different Database states and probabilistic transitions correspond to application of probabilistic rules (see details in Section II).

Demonstration Scenario: We demonstrate the operation of Trivia Masster on a real-life Database of facts, collected from users of the GUESS What game that was developed on top of the IBM Crowd Computing Gaming Platform in IBM Haifa Research Labs. The *facts* in this demonstration provide information on capital cities in the world. The demonstration considers users that had already played with the system

This work was partly supported by the EU project Mancoosi, by the Israel Science Foundation and by the US-Israel Binational Science Foundation.

Country	Capital	User
China	Beijing	Alice
Netherlands	Amsterdam	Bob
Netherlands	Hague	Carol
China	Shanghai	Dan

(a) Capitals

User	Reliability
Alice	5
Bob	7
Carol	9
Dan	12

(b) Users

Fig. 1: Users and Capitals Tables

("known" users), as well as new users (volunteers from the demonstration audience). We start with two known users: a "reliable" user (whose answers so far were mostly determined to be correct), and an "unreliable" user. We will login into the system using the accounts of these two users, show the Trivia questions presented to them, and the manner in which (in)correct responses affect both their score as well as the Database state. For the latter, we will ask the system to *answer (relevant) questions*, and show how the answers depend on the users input and score. Next, we consider new (volunteer) users. We first let them play Trivia for a short while, showing how the system determines their reliability (reflected by their score). Then, as above, we show how their input affects the database state. To conclude the demonstration we allow the audience to look "under the hood": We show the declarative Data Cleaning rules and explain how they serve each of the system components. We further show how the change of rules supports different cleaning policies. Finally, we focus on one of the questions that were previously posed to the Question Answering component, and show "snapshots" (i.e. a sample of the Database states, restricted to the facts of interest) of the cleaning process performed by the Markov Chain Monte Carlo Algorithm, for answering the question.

II. Technological Background

The *Trivia Masster* system manages its data in a Relational Database. The core of the system is a declarative framework for specifying (and recursively executing) probabilistic data cleaning rules over this data. Rules are defined using relational algebra enriched with a special operator called *repair-key* [10], used to introduce probabilities. We briefly and intuitively explain this below, using as an example a simple PageRank-style cleaning policy. The full technical details appear in [8].

Database: Assume that at a certain point of time, the database contains the relations $Capitals$ and $Users$ depicted in Table 1. The former includes information regarding the capital cities of various countries and the name of the user that submitted each fact; the latter includes for each user a numerical value reflecting the user's reliability (we explain below how are these values computed).

Repair-Key: The primary key in $Capitals$ is the attribute $Country$; but note that there exists contradictions (i.e. primary key violations) in this relation. We can solve these contradictions in a probabilistic way, with the probability of each choice be proportional to its support, i.e. to the reliability of users submitting the fact (relative to the reliability of users submitting contradicting facts). For that, we can first join $Capitals$ and $Users$, to obtain the support of each fact, then apply the *repair-key* construct [10] over the joined table:

$$\text{repair-key}_{Country@Reliability}(Capitals \bowtie Users)$$

This means that we make a probabilistic choice for a single capital for each country (thus "cleaning" the table), with probabilities dictated by the relative reliability of the users proposing this capital. The result of applying repair-key is a set of possible worlds (databases), each being a different possible clean instance where every country bears a single capital; each such world is associated with a probability value, which corresponds to the proportional support value in its facts.

Recursive application of Probabilistic Update Rules: After applying the above rule we have in hand a set of possible worlds for the $Capitals$ relation, each containing a clean instance and associated with some probability. For every such world, we may choose to update our trust in users based upon their contribution to the currently clean Database. For that, we define an update rule for the $Users$ relation that is the result of a query that counts the number of facts given by each user. This query will be applied in every possible world. Now, again in every possible world, given the updated reliability values of users, we can re-clean the original $Capitals$ relation (that contains contradictions) using the first query. This results in the creation of new worlds; in each such world we can re-apply the second query to update our assessment of the users reliability, and so on. This way, we in fact let the probabilistic update rules (queries) induce a traversal over a *Markov Chain* (MC) [11]. The MC states are possible database instances and its transitions (and their corresponding probabilities) are dictated by the rules.

Query Semantics: Consider a query q over the Database, e.g. a selection query that asks for the capital of England. Consider also the Data Cleaning rules, and a boolean predicate indicating that a given Database is clean (for our example, by checking for primary key violations in $Capitals$). The output is then defined to consist of all tuples that appear in the evaluation of q over some clean instance of the Database; each such tuple is associated with a number, which is the sum of probabilities of all clean instances D for which the tuple appears in the result of applying q over D. Computing this proportion directly is of course infeasible [8], thus we provide a sampling algorithm as explained next.

Query Evaluation: We use the algorithm of [8] for query evaluation. Recall the analogy between our probabilistic update rules and Markov Chains. In principal, we can use for query evaluation a *Markov Chain Monte Carlo* (MCMC) algorithm, performing a random walk and evaluating the query event on each Database state traversed along the walk, and then reporting the observed proportion of each query answer as an approximation for its probability. Note, however that MCMC algorithms are guaranteed to output good approximations only when the underlying Markov Chain is ergodic [9]. In our case, the MC underlying the Data Cleaning rules set is ergodic if it is strongly connected. We handle the case where it is not strongly connected by computing the strongly connected components and employing an MCMC algorithm separately for each component, then combining the results (see [8] for details). One important practical issue is when to stop this sampling process. We show in [8], using theoretical results on Markov Chains, that the probabilities obtained during this sampling process are guaranteed to converge. Our stop condition thus examines the

difference in-between two consecutively obtained samples, and halts when (for a configurable number of consecutive steps) this difference is at most some small configurable ϵ.

Trivia Masster - Query Answering, Scoring, and Question Selection: We next explain how the underlying system of probabilistic and recursive rules can be used for implementation of the required features of *Trivia Masster*. We use a rules set as depicted above, implementing a PageRank-style cleaning process, and change the query event according to our goal. Specifically, for query answering, the query event is simply the translation of the form-based query asked by the user, into a relational algebra query. For the scoring function computation, the query event asks for the reliability values of users, and then compute a weighted average of the result with the result of a non-probabilistic query asking for the *novelty* of the facts submitted by the user (i.e. how many users have submitted the same facts before the current user).

Related Work: Crowdsourcing is an emerging paradigm that harnesses mass of users to construct knowledge bases and to perform various types of tasks. Crowdsourcing applications have been recognized as a powerful tool for collecting the wisdom of the crowd in various domains, e.g. distribution of programming tasks [12] or performing tasks that require human intelligence tasks [13]. The usage of *games* to address difficult tasks was proposed in [2] for various tasks, such as object recognition in images and collection of user preferences. We have mentioned in the Introduction the specific challenges in our context and the novelty of our framework, relative to existing data cleaning techniques. We further note that a declarative framework for probabilistic rules, based on Markov Chains was suggested in [7], but did not allow definition of recursive rules, hence for example cannot express the PageRank-style cleaning rules depicted above.

III. System Overview

Trivia Masster is implemented in C# and PHP, uses the MySQL Database and a Flash-based GUI. Figure 2 depicts the system architecture. At a high-level, the system design includes three main components: the first one is *GUESS What*, being the interface for users that *contribute facts* to the Database; the second component is a *Question Answering* service that allows users to ask questions (and obtain answers) based on the data collected by GUESS What. These two components use a third component, the *Evaluation Engine*, which is in charge of maintaining and cleaning a relational Database of collected facts. It provides answers to user questions and is responsible for assigning scores to users of GUESS What and deciding which questions to present to them. Following is a brief overview of these three components and their interaction.

GUESS What: The GUESS What module is responsible for *data collection*. It was developed in IBM Haifa Research Labs on top of the IBM Crowd Computing Gaming Platform, which facilitates the development of games played by a mass of users. GUESS What is connected to the other components of Trivia Masster, to allow users to ask questions over the data that was collected during the game.

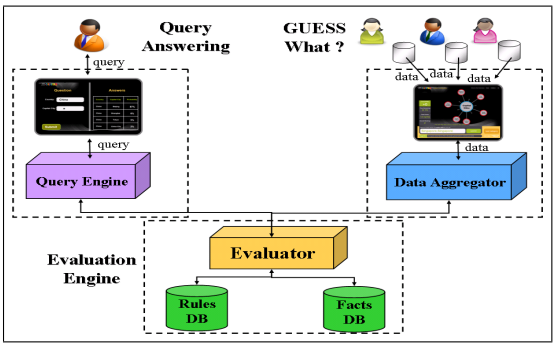

Fig. 2: System Architecture

The User Interface of GUESS What is Flash-based, and its underlying backend is PHP-based and is deployed on an Apache application server. Users are presented with Trivia questions in various areas and may then provide one or more answers. The User Interface of GUESS What is depicted in Fig. 3. The circle appearing in the middle is the *question*, which in the given example asks for "Capital Cities". The circles surrounding it are answers provided by the current player, in the given example corresponding to couples of $(Country, Capital)$. Users are also given hints on how many items were contributed by other users, without seeing them explicitly (done for motivation purposes). For questions with multiple possible answers, the user is asked to provide as many answers as she can, in a limited time frame. The user is assigned *points* based on the *novelty* and *correctness* of her submitted facts; the rank of these two properties is achieved by evaluating two corresponding queries over the database, using the Evaluation Engine module depicted below. As described in Section II, testing for the correctness of a given fact amounts to using the fact as a query event and computing the probability that it holds in a random database generated by the recursive application of the data cleaning rules. The higher this probability is, the more points the users are given. As for novelty, this corresponds to a simple query, testing whether the number of users that had already submitted the fact exceeds a certain threshold.

Fig. 3: GUESS What User Interface

Question Answering: The Question Answering component allows users to ask questions on the data collected by GUESS What. It presents forms for querying the data collected during the Trivia game. An example for such form is

Fig. 4: Question Answering User Interface

presented in Figure 4, where the presented form allows to ask which is the capital of a given country (in this example, "what is the capital of China?"). The filled-in form is then compiled into a relational algebra query. The query is submitted to the Evaluation Engine , and the (top-k) available answers, along with their probability of being correct, are presented to the user.

Evaluation Engine: The Evaluation Engine module is the core component of Trivia Masster. It maintains a *Facts Database*, that stores facts collected from users, as well as information regarding the profiles of users (their reliability, number of correct submitted facts, etc.), and a *Rules Database* that stores data cleaning rules, to be applied over the collected facts. As mentioned above, these rules are written in relational algebra enriched with a repair-key construct. They are executed recursively using a Markov Chain Monte Carlo algorithm, to answer queries submitted by the Question Answering or the GUESS What modules. The declarative nature of these rules allows to easily modify and adapt them to different Data Cleaning strategies.

IV. Demonstration Scenario

We demonstrate the operation and capabilities of Trivia Masster on a real-life Database of facts depicting capital cities, constructed in IBM by allowing users to play GUESS What. The Database contains correct and incorrect facts, and in particular some of the players are "reliable", meaning that they mostly provide correct answers, while others are unreliable and most of their answers were determined to be incorrect. GUESS What presents to users questions related to data items for which it has little information or confidence, in order to gain knowledge and certainty w.r.t. these items, as well as questions on items for which it has a higher confidence, in order to derive user trust information and user scores.

The demonstration will consist of two monitors, one of them showing the GUESS What interface and the second showing the Question Answering interface. It will begin as we login into GUESS What as a particular player which is reliable, and show the Trivia questions presented to her; before answering the Trivia questions, we pose them to the Question Answering component, and observe that the questions presented to her are indeed the ones with unknown or low confidence answers. We then start answering the presented Trivia questions in GUESS What , and again submit these questions to Question Answering; we will observe the effect that the reliable user has on the order of answers in Question Answering, as well as on their probability values. For questions in which the system has a high confidence, we deliberately answer some of the questions incorrectly, and observe that this yields a decrease in the user reliability that in turn leads to a lesser effect on answers presented by the system. To show an extreme case of this, we log-in as a different user, known from her previous answers to be unreliable, and repeat the demonstration this time the user answers will hardly affect the results presented in the Question Answering interface.

To further show the creation of user profiles, and the interplay between user reliability and the database content, we invite a volunteer from the audience to use the system. We first let her play Trivia for a short while, and in parallel we use the Question Answering interface to show the effect of her answers on the Database. We show that her first answers bear an insignificant effect the answers given by the system, but depending on the correctness of her results, she may gain an increasingly significant effect on these answers.

Finally, we allow the audience to look "under the hood" of the system. We first present the Data Cleaning rules that were declaratively specified beforehand, and were used throughout the demonstration. As mentioned above, these rules are written in relational algebra enriched with the repair-key operator, and we use these rules as examples to explain the use and effect of repair-key; specifically, we explain how simple modifications of the rules may allow to support completely different Data Cleaning policies. We then show the operation of these rules, focusing on the evaluation process of a question we presented, in the first part of the demonstration, to Question Answering. This evaluation process is shown as a sampled series of 5 significant Database instance states obtained while evaluating the Markov Chain Monte Carlo Algorithm, restricted to the facts of interest. We explain the transitions and the computation of answers that were eventually obtained.

References

[1] D. C. Brabham, "Crowdsourcing as a Model for Problem Solving: An Introduction and Cases," *Convergence*, vol. 14, no. 1, pp. 75–90, 2008. [Online]. Available: http://con.sagepub.com/cgi/content/abstract/14/1/75
[2] L. von Ahn and L. Dabbish, "Designing games with a purpose," *Commun. ACM*, vol. 51, no. 8, pp. 58–67, 2008.
[3] H. Ma, R. Chandrasekar, C. Quirk, and A. Gupta, "Improving search engines using human computation games," in *CIKM '09*.
[4] Q. Su, D. Pavlov, J.-H. Chow, and W. C. Baker, "Internet-scale collection of human-reviewed data," in *WWW '07*.
[5] A. Arasu, S. Chaudhuri, and R. Kaushik, "Learning string transformations from examples," *PVLDB*, vol. 2, no. 1, 2009.
[6] A. Galland, S. Abiteboul, A. Marian, and P. Senellart, "Corroborating information from disagreeing views," in *WSDM '10*.
[7] R. Jampani et al., "Mcdb: a monte carlo approach to managing uncertain data," in *SIGMOD '08*.
[8] D. Deutch, C. Koch, and T. Milo, "On probabilistic fixpoint and markov chain query languages," in *PODS '10*.
[9] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., 2005.
[10] C. Koch, "Approximating predicates and expressive queries on probabilistic databases," in *PODS*, 2008.
[11] D. Freedman, *Markov Chains*. Springer-Verlag, 1983.
[12] "Top coder," http://www.topcoder.com/.
[13] "Amazon's mechanical turk," https://www.mturk.com/.