How Do Humans and Data Systems Establish a Common Query Language?

Ben McCamish
Oregon State University
mccamisb@oregonstate.edu

Vahid Ghadakchi
Oregon State University
ghadakcv@oregonstate.edu

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

Liang Huang
Oregon State University
liang.huang@oregonstate.edu

Behrouz Touri
University of CA San Diego
btouri@eng.ucsd.edu

ABSTRACT

As most users do *not* precisely know the structure and/or the content of databases, their queries do *not* exactly reflect their information needs. While database management systems (DBMS) may interact with users and use their feedback on the returned results to learn the information needs behind their queries, current query interfaces assume that users do *not* learn and modify the way way they express their information needs in form of queries during their interaction with the DBMS. Using a real-world interaction workload, we show that users learn and modify how to express their information needs during their interactions with the DBMS and their learning is accurately modeled by a well-known reinforcement learning mechanism. As current data interaction systems assume that users do *not* modify their strategies, they cannot discover the information needs behind users' queries effectively. We model the interaction between users and DBMS as a game with identical interest between two rational agents whose goal is to establish a common language for representing information needs in form of queries. We propose a reinforcement learning method that learns and answers the information needs behind queries and adapts to the changes in users' strategies and prove that it stochastically improves the effectiveness of answering queries. We propose two efficient implementation of this method over large relational databases. Our empirical studies over real-world query workloads indicate that our algorithms are efficient and effective.

1. INTRODUCTION

Most users do not know the structure and content of databases and concepts such as schema or formal query languages sufficiently well to express their information needs precisely in the form of queries [8]. They may convey their intents in easy-to-use but inherently ambiguous forms, such as keyword queries, which are open to numerous interpretations. Thus, it is very challenging for a database management system (DBMS) to understand and satisfy the intents behind these queries. The fundamental challenge in the interaction of these users and DBMS is that the users and DBMS represent intents in different forms.

Many such users may explore a database to find answers for various intents over a rather long period of time. For these users, database querying is an inherently interactive and continuous process. As both the user and DBMS have the same goal of the user receiving her desired information, the user and DBMS would like to gradually improve their understanding of each other and reach a *common language of representing intents* over the course of various queries and interactions. The user may learn more about the structure and content of the database and how to express intents as she submits queries and observes the returned results. Also, the DBMS may learn more about how the user expresses her intents by leveraging user feedback on the returned results. The user feedback may include clicking on the relevant answers [33], or the signals sent in touch-based devices [20]. Ideally, the user and DBMS should establish as quickly as possible this common representation of intents in which the DBMS accurately understands all or most user's queries.

Researchers have developed systems that leverage user feedback to help the DBMS understand the intent behind ill-specified and vague queries more precisely [6]. These systems, however, generally assume that a user does *not* modify her method of expressing intents throughout her interaction with the DBMS. For example, they maintain that the user picks queries to express an intent according to a fixed probability distribution. It is known that the learning methods that are useful in a static setting do not deliver desired outcomes in a setting where all agents may modify their strategies [14]. Hence, one may not be able to use current techniques to help the DBMS understand the users' information need in a rather long-term interaction.

To the best of our knowledge, the impact of user learning on database interaction has been generally ignored. In this paper, we propose a novel framework that formalizes the interaction between the user and the DBMS as a game with identical interest between two active and potentially rational agents: the user and DBMS. The common goal of the user and DBMS is to reach a mutual understanding on expressing information needs in the form of keyword queries. In each interaction, the user and DBMS receive certain payoffs according to how much the returned results are relevant

to the intent behind the submitted query. The user receives her payoff by consuming the relevant information and the DBMS becomes aware of its payoff by observing the user's feedback on the returned results. We believe that such a game-theoretic framework naturally models the long-term interaction between the user and DBMS. We explore the user learning mechanisms and propose algorithms for the DBMS to improve its understanding of intents behind the user queries effectively and efficiently over large databases. In particular, we make the following contributions:

- We model the long term interaction between the user and DBMS using keyword queries as a particular type of game called a signaling game [9] in Section 2.
- Using extensive empirical studies over a real-world interaction log, we show that users modify the way they express their information need over their course of interactions in Section 3. We also show that this adaptation is accurately modeled by a well-known reinforcement learning algorithm [27] in experimental game-theory.
- We describe our data interaction system that provides an efficient implementation of our reinforcement learning method on large relational databases in Section 5. In particular, we first propose an algorithm that implements our learning method called *Reservoir*. Then, using certain mild assumptions and the ideas of sampling over relational operators, we propose another algorithm called *Poisson-Olken* that implements our reinforcement learning scheme and considerably improves the efficiency of *Reservoir*.
- We report the results of our empirical studies on measuring the effectiveness of our reinforcement learning method and the efficiency of our algorithms using real-world and large interaction workloads, queries, and databases in Section 6. Our results indicate that our proposed reinforcement learning method is more effective than the start-of-the-art algorithm for long-term interactions. They also show that *Poisson-Olken* can process queries over large databases faster than the *Reservoir* algorithm.

2. A GAME-THEORETIC FRAMEWORK

Users and DBMSs typically achieve a common understanding *gradually* and using a *querying/feedback* paradigm. After submitting each query, the user may revise her strategy of expressing intents based on the returned result. If the returned answers satisfy her intent to a large extent, she may keep using the same query to articulate her intent. Otherwise, she may revise her strategy and choose another query to express her intent in the hope that the new query will provide her with more relevant answers. We will describe this behavior of users in Section 3 in more detail. The user may also inform the database system about the degree by which the returned answers satisfy the intent behind the query using explicit or implicit feedback, e.g., click-through information [13]. The DBMS may update its interpretation of the query according to the user's feedback.

Intuitively, one may model this interaction as a game between two agents with identical interests in which the agents communicate via sharing queries, results, and feedback on the results. In each interaction, both agents will receive some reward according to the degree by which the returned result for a query matches its intent. The user receives her rewards in the form of answers relevant to her intent and the DBMS receives its reward through getting positive feedback on the returned results. The final goal of both agents is to maximize the amount of reward they receive during the course of their interaction.

2.1 Intent

An *intent* represents an information need sought after by the user. Current keyword query interfaces over relational databases generally assume that each intent is a query in a sufficiently expressive query language in the domain of interest, e.g., Select-Project-Join subset of SQL [8, 18]. Our framework and results are orthogonal to the language that precisely describes the users' intents. Table 1 illustrates a database with schema *Univ(Name, Abbreviation, State, Rank)* that contains information about university rankings. A user may want to find the information about university *MSU* in Michigan, which is precisely represented by the intent e_2 in Table 2(a), which using the Datalog syntax [1] is: $ans(z) \leftarrow Univ(x, `MSU', `MI', z)$.

2.2 Query

Users' articulations of their intents are *queries*. Many users do not know the formal query language, e.g., SQL, that precisely describes their intents. Thus, they may prefer to articulate their intents in languages that are easy-to-use, relatively less complex, and ambiguous such as keyword query language [18, 8]. In the proposed game-theoretic frameworks for database interaction, we assume that the user expresses her intents as keyword queries. More formally, we fix a countably infinite set of terms, i.e., keywords, T. A *keyword query* (query for short) is a nonempty (finite) set of terms in T. Consider the database instance in Table 1. Table 2 depicts a set of intents and queries over this database. Suppose the user wants to find the information about Michigan State University in Michigan, i.e. the intent e_2. Because the user does not know any formal database query language and may not be sufficiently familiar with the content of the data, she may express intent e_2 using q_2 : *'MSU'*.

Some users may know a formal database query language that is sufficiently expressive to represent their intents. Nevertheless, because they may not know precisely the content and schema of the database, their submitted queries may not always be the same as their intents [6]. For example, a user may know how to write a SQL query. But, since she may not know the state abbreviation *MI*, she may articulate intent e_2 as $ans(z) \leftarrow Univ(x, `MSU', y, z)$, which is different from e_2. We plan to extend our framework for these scenarios in future work. But, in this paper, we assume that users articulate their intents as keyword queries.

2.3 User Strategy

The user strategy indicates the likelihood that the user submits query q given that her intent is e. In practice, a user has finitely many intents and submits finitely many queries in a finite period of time. Hence, we assume that the sets of the user's intents and queries are finite. We index each user's intent and query by $1 \leq i \leq m$ and $1 \leq j \leq n$, respectively. A user strategy, denoted as U, is a $m \times n$ row-stochastic matrix from her intents to her queries. The matrix on the top of Table 3(a) depicts a user strategy using intents and queries in Table 2. According to this strategy, the user submits query q_2 to express intents e_1, e_2, and e_3.

Table 1: A database instance of relation Univ

Name	Abbreviation	State	Rank
Missouri State University	MSU	MO	20
Mississippi State University	MSU	MS	22
Murray State University	MSU	KY	14
Michigan State University	MSU	MI	18

Table 2: Intents and Queries

2(a) Intents

Intent#	Intent
e_1	$ans(z) \leftarrow Univ(x, `MSU', `MS', z)$
e_2	$ans(z) \leftarrow Univ(x, `MSU', `MI', z)$
e_3	$ans(z) \leftarrow Univ(x, `MSU', `MO', z)$

2(b) Queries

Query#	Query
q_1	'MSU MI'
q_2	'MSU'

Table 3: Two strategy profiles over the intents and queries in Table 2. User and DBMS strategies at the top and bottom, respectively.

3(a) A strategy profile

	q_1	q_2
e_1	0	1
e_2	0	1
e_3	0	1

	e_1	e_2	e_3
q_1	0	1	0
q_2	0	1	0

3(b) Another strategy profile

	q_1	q_2
e_1	0	1
e_2	1	0
e_3	0	1

	e_1	e_2	e_3
q_1	0	1	0
q_2	0.5	0	0.5

2.4 DBMS Strategy

The DBMS interprets queries to find the intents behind them. It usually interprets queries by mapping them to a subset of SQL [8, 16]. Since the final goal of users is to see the result of applying the interpretation(s) on the underlying database, the DBMS runs its interpretation(s) over the database and returns its results. Moreover, since the user may *not* know SQL, suggesting possible SQL queries may not be useful. A DBMS may *not* exactly know the language that can express all users' intents. Current usable query interfaces, including keyword query systems, select a query language for the interpreted intents that is sufficiently complex to express many users' intents and is simple enough so that the interpretation and running its outcome(s) are done efficiently [8].

To better leverage users feedback during the interaction, the DBMS must show the results of and get feedback on a sufficiently diverse set of interpretations [15, 31]. Of course, the DBMS should ensure that this set of interpretations is relatively relevant to the query, otherwise the user may become discouraged and give up querying. This dilemma is called the *exploitation versus exploration* trade-off. A DBMS that only *exploits*, returns top-ranked interpretations according to its scoring function. Hence, the DBMS may adopt a *stochastic strategy* to both exploit and explore: it randomly selects and shows the results of intents such that the ones with higher scores are chosen with larger probabilities [15, 31]. In this approach, users are mostly shown results of interpretations that are relevant to their intents according to the current knowledge of the DBMS and provide feedback on a relatively diverse set of interpretations. More formally, given Q is a set of all keyword queries, the DBMS strategy D is a stochastic mapping from Q to L, where L is some interpretation of the keyword query that contains some tuples from the underlying database. The matrix on the bottom of Table 3(a) depicts a DBMS strategy for the intents and queries in Table 2. Based on this strategy, the DBMS uses a exploitative strategy and always interprets query q_2 as e_2. The matrix on the bottom of Table 3(b) depicts another DBMS strategy for the same set of intents and queries. In this example, DBMS uses a randomized strategy and does both exploitation and exploration. For instance, it explores e_1 and e_3 to answer q_2 with equal probabilities, but it always returns e_2 in the response to q_1.

2.5 Interaction & Adaptation

The data interaction game is a repeated game with identical interest between two players, the user and the DBMS. At each round of the game, i.e., a single interaction, the user selects an intent according to the prior probability distribution π. She then picks the query q according to her strategy and submits it to the DBMS. The DBMS observes q and interprets q based on its strategy, and returns the results of the interpretation(s) on the underlying database to the user. The user provides some feedback on the returned tuples and informs the DBMS how relevant the tuples are to her intent. In this paper, we assume that the user informs the DBMS if some tuples satisfy the intent via some signal, e.g., selecting the tuple, in some interactions.

Next, we compute the expected payoff of the players. Since DBMS strategy D maps each query to a finite set of interpretations, and the set of submitted queries by a user, or a population of users, is finite, the set of interpretations for all queries submitted by a user, denoted as L^s, is finite. Hence, we show the DBMS strategy for a user as an $n \times o$ row-stochastic matrix from the set of the user's queries to the set of interpretations L^s. We index each interpretation in L^s by $1 \leq \ell \leq o$. Each pair of the user and the DBMS strategy, (U,D), is a *strategy profile*. The expected payoff for both players with strategy profile (U,D) is as follows, where $r(e_i, e_\ell)$ is some effectiveness metric such as *precision at k* [21].

$$u_r(U, D) = \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} U_{ij} \sum_{\ell=1}^{o} D_{j\ell}\ r(e_i, e_\ell), \qquad (1)$$

The expected payoff reflects the degree by which the user and DBMS have reached a common language for communication. This value is high for the case in which the user knows which queries to pick to articulate her intents and the DBMS returns the results that satisfy the intents behind the user's queries. Hence, this function reflects the success of the communication and interaction. For example, given that all intents have equal prior probabilities, intuitively, the strategy profile in Table 3(b) shows a larger degree of mutual understanding between the players than the one in Table 3(a). This is reflected in their values of expected payoff as the expected payoffs of the former and latter are $\frac{2}{3}$ and $\frac{1}{3}$, respectively. We note that the DBMS may *not* know the set of users' queries beforehand and does *not* compute the expected payoff directly. Instead, it uses query answering algorithms that leverage user feedback, such that the expected payoff improves over the course of several interactions.

3. USER LEARNING MECHANISM

It is well established that humans show reinforcement behavior in learning [29, 24]. Many lab studies with human subjects conclude that one can model human learning using reinforcement learning models [29, 24]. The exact reinforcement learning method used by a person, however, may vary based on her capabilities and the task at hand. We have performed an empirical study of a real-world interaction log to find the reinforcement learning method(s) that best explain the mechanism by which users adapt their strategies during interaction with a DBMS.

3.1 Human Learning Schemes

To provide a comprehensive comparison, we evaluate six reinforcement learning methods used to model human learning in experimental game theory and/or Human Computer Interaction (HCI) [27, 5]. These methods mainly vary based on 1) the degree by which the user considers past interactions when computing future strategies, 2) how they update the user strategy, and 3) the rate by which they update the user strategy. *Win-Keep/Lose-Randomize* keeps a query with non-zero reward in past interactions for an intent. If such a query does not exist, it picks a query randomly. *Latest-Reward* reinforces the probability of using a query to express an intent based on the most recent reward of the query to convey the intent. *Bush and Mosteller's* and *Cross's* models increases (decreases) the probability of using a query based its past success (failures) of expressing an intent. A query is successful if it delivers a reward more than a given threshold, e.g., zero. *Roth and Erev's* model uses the aggregated reward from past interactions to compute the probability by which a query is used. *Roth and Erev's modified* model is similar to Roth and Erev's model, with an additional parameter that determines to what extent the user *forgets* the reward received for a query in past interactions.

3.2 Empirical Analysis

Interaction Logs: We use an anonymized Yahoo! interaction log for our empirical study, which consists of keyword queries submitted to a Yahoo! search engine in July 2010 [32]. We have used three different contiguous subsamples of this log whose information is shown in Table 4. The duration of each subsample is the time between the timestamp of the first and last interaction records. The records of the 8H-interaction sample appear at the beginning of the the 43H-interaction sample, which themselves appear at the beginning of the 101H-interaction sample.

Intent & Reward: Accompanying the interaction log is a set of *relevance judgment scores* for each query and result pair. Each relevance judgment score is a value between 0 and 4 and shows the degree of relevance of the result to the query, with 0 meaning not relevant at all and 4 meaning the most relevant result. We define the intent behind each query as the set of results with non-zero relevance scores. We use the standard ranking quality metric Normalized Discounted Cumulative Gain (NDCG) for the returned results of a query as the reward in each interaction as it models different levels of relevance [21]. The value of NDCG is between 0 and 1 and it is 1 for the most effective list.

Training & Testing: We have used a set of 5,000 records that appear in the interaction log immediately before the first subsample of Table 4 and found the optimal values for parameters using grid search and the sum of squared errors. We train and test a single user strategy over each subsample and model, which represents the strategy of the user population in each subsample. After estimating parameters, we train the user strategy using each model over 90% of the total number of records in each selected subsample in the order by which the records appear in the interaction log and test over the remaining 10% using the user strategy computed at the end of the training phase. We report the mean squared errors over all intents in the testing phase for each subsample and model in Table 5. A lower mean squared error implies that the model more accurately represents the users' learning method. We have excluded the Latest Reward results from the figure as they are an order of magnitude worse than the others.

Table 4: Subsamples of Yahoo! interaction log

Duration	#Interactions	#Users	#Queries	#Intents
˜8H	622	272	111	62
˜43H	12323	4056	341	151
˜101H	195468	79516	13976	4829

Table 5: Accuracies of learning over the subsamples of Table 4

Methods	Duration		
	101H	43H	8H
Bush and Mosteller's	0.0672	0.1880	0.2434
Cross's	0.0686	0.1908	0.2472
Roth and Erev's	**0.0666**	**0.1827**	0.2522
Roth and Erev's Modified	**0.0666**	**0.1827**	0.2522
Win-Keep/Lose-Randomize	0.0713	0.1876	**0.2364**

Results: Win-Keep/Lose-Randomize performs surprisingly more accurately than other methods for the 8H-interaction subsample. It indicates that in short-term and/or beginning of their interactions, users may not have enough interactions to leverage a more complex learning scheme and use a rather simple mechanism to update their strategies. Both Roth and Erev's methods use the accumulated reward values to adjust the user strategy gradually. Hence, they cannot precisely model user learning over a rather short interaction and are less accurate than relatively more aggressive learning models such as Bush and Mosteller's and Cross's over this subsample. Both Roth and Erev's deliver the same result and outperform other methods in the 43-H and 101-H subsamples. Win-Keep/Lose-Randomize is the least accurate method over the two larger subsamples. Since larger subsamples provide more training data, the predication accuracy of all models improves as the interaction subsamples becomes larger. The learned value for the *forget* parameter in the Roth and Erev's modified model is very small and close to zero in our experiments, therefore, it generally acts like the Roth and Erev's model.

Long-term communications between users and DBMS may include multiple sessions. Since Yahoo! query workload contains the time stamps and user ids of each interaction, we have been able to extract the starting and ending times of each session. Our results indicate that as long as the user and DBMS communicate over sufficiently many of interactions, e.g., about 10k for Yahoo! query workload, the users follow the Roth and Erev's model of learning. Given that the communication of the user and DBMS involve sufficiently

many interactions, we have *not* observed any difference in the mechanism by which users learn based on the numbers of sessions in the user and DBMS communication.

Conclusion: Our analysis indicates that users show a substantially intelligent behavior when adopting and modifying their strategies over relatively medium and long-term interactions. They leverage their past interactions and their outcomes, i.e., have an effective long-term memory. This behavior is most accurately modeled using Roth and Erev's model. Hence, in the rest of the paper, we set the user learning method to this model.

4. LEARNING ALGORITHM FOR DBMS

Current systems generally assume that a user does *not* learn and/or modify her method of expressing intents throughout her interaction with the DBMS. However, it is known that the learning methods that are useful in static settings do not deliver desired outcomes in the dynamic ones [3]. Moreover, it has been shown that if the players do *not* use the right learning algorithms in games with identical interests, the game and its payoff may not converge to any desired states [28]. Thus, choosing the correct learning mechanism for the DBMS is crucial to improve the payoff and converge to a desired state.

4.1 DBMS Reinforcement Learning

We adopt Roth and Erev's learning method for adaptation of the DBMS strategy, with a slight modification. The original Roth and Erev method considers only a single action space. In our work, this would translate to having only a single query. Instead we extend this such that each query has its own action space or set of possible intents. The adaptation happens over discrete time $t = 0, 1, 2, 3, \ldots$ instances where t denotes the tth interaction of the user and the DBMS. We refer to t simply as the iteration of the learning rule. For simplicity of notation, we refer to intent e_i and result s_ℓ as intent i and ℓ, respectively, in the rest of the paper. Hence, we may rewrite the expected payoff for both user and DBMS as:

$$u_r(U, D) = \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} U_{ij} \sum_{\ell=1}^{o} D_{j\ell} r_{i\ell},$$

where $r : [m] \times [o] \to \mathbb{R}^+$ is the effectiveness measure between the intent i and the result, i.e., decoded intent ℓ. With this, the reinforcement learning mechanism for the DBMS adaptation is as follows.

a. Let $R(0) > 0$ be an $n \times o$ initial reward matrix whose entries are strictly positive.

b. Let $D(0)$ be the initial DBMS strategy with $D_{j\ell}(0) = \frac{R_{j\ell}(0)}{\sum_{\ell=1}^{o} R_{j\ell}(0)} > 0$ for all $j \in [n]$ and $\ell \in [o]$.

c. For iterations $t = 1, 2, \ldots$, do

   i. If the user's query at time t is $q(t)$, DBMS returns a result $E(t) \in E$ with probability:

   $$P(E(t) = i' \mid q(t)) = D_{q(t)i'}(t).$$

   ii. User gives a reward $r_{ii'}$ given that i is the intent of the user at time t. Note that the reward depends both on the intent i at time t and the result i'. Then, set

   $$R_{j\ell}(t+1) = \begin{cases} R_{j\ell}(t) + r_{i\ell} & \text{if } j = q(t) \text{ and } \ell = i' \\ R_{j\ell}(t) & \text{otherwise} \end{cases}. \quad (2)$$

   iii. Update the DBMS strategy by

   $$D_{ji}(t+1) = \frac{R_{ji}(t+1)}{\sum_{\ell=1}^{o} R_{j\ell}(t+1)}, \quad (3)$$

   for all $j \in [n]$ and $i \in [o]$.

In the above algorithm $R(t)$ is simply the reward matrix at time t. We have also proved the following:

THEOREM 4.1. *The proposed learning algorithm in Section 4.1 converges almost surely when the user learns using Roth and Erev's model.*

The above result implies that the effectiveness of the DBMS, stochastically speaking, increases as time progresses when the learning rule in Section 4.1 is utilized. The user may also learn at a relatively slow rate such that from the perspective of the database it seems as though the user isn't learning. Of course, the user may not perform any learning. Our results also hold for the case when the user doesn't learn. We have also proved that the payoff of the two agents only increases or remains the same. To see all proofs in full, we refer the reader to our published work [22].

It is quite costly to materialize and maintain the strategy of the DBMS as shown in the previous examples. Thus, we maintain the strategy and reinforcements in a constructed feature space using n-grams for each attribute value. Our mapping is then a mapping from query features to tuple features.

5. EFFICIENT QUERY ANSWERING OVER RELATIONAL DATABASES

An efficient implementation of our algorithm proposed in Section 4 over large relational databases poses two challenges. First, since the set of possible interpretations and their results for a given query is enormous, one has to find efficient ways of maintaining users' reinforcements and updating DBMS strategy. Second, keyword and other usable query interfaces over databases normally return the top-k tuples according to some scoring functions [16, 8]. Due to a series of seminal works by database researchers [12], there are efficient algorithms to find such a list of answers. Nevertheless, our reinforcement learning algorithm uses a randomized semantics for answering algorithms in which candidate tuples are associated a probability for each query that reflects the likelihood by which it satisfies the intent behind the query. The tuples must be returned randomly according to their associated probabilities. Using (weighted) sampling to answer SQL queries with aggregation functions approximately and efficiently is an active research area [17]. However, there has not been any attempt on using a randomized strategy to answer so-called point queries over relational data and achieve a balanced exploitation-exploration trade-off efficiently.

5.1 Keyword Query Interface

We use the current architecture of keyword query interfaces over relational databases that directly use schema information to interpret the input keyword query [8]. A notable example of such systems is IR-Style [16]. We provide an overview of the basic concepts of such a system. We refer the reader to [16, 8] for more explanation.

Tuple-set: Given keyword query q, a *tuple-set* is a set of tuples in a base relation that contain some terms in q. After receiving q, the query interface uses an inverted index to compute a set of tuple-sets. For instance, consider a database of products with relations *Product(pid, name)*, *Customer(cid, name)*, and *ProductCustomer(pid, cid)* where *pid* and *cid* are numeric strings. Given query *iMac John*, the query interface returns a tuple-set from *Product* and a tuple-set from *Customer* that match at least one term in the query.

Candidate Network: A *candidate network* is a join expression that connects the tuple-sets via primary key-foreign key relationships. A candidate network joins the tuples in different tuple-sets and produces joint tuples that contain the terms in the input keyword query. One may consider the candidate network as a join tree expression whose leaves are tuple-sets. For instance, one candidate network for the aforementioned database of products is *Product* $\bowtie$ *ProductCustomer* $\bowtie$ *Customer*. To connect tuple-sets via primary key-foreign key links, a candidate network may include base relations whose tuples may not contain any term in the query, e.g., *ProductCustomer* in the preceding example. Given a set of tuple-sets, the query interface uses the schema of the database and progressively generates candidate networks that can join the tuple-sets. For efficiency considerations, keyword query interfaces limit the number of relations in a candidate network to be lower than a given threshold. Keyword query interfaces normally compute the score of joint tuples by summing up the scores of their constructing tuples multiplied by the inverse of the number of relations in the candidate network to penalize long joins [8]. We use the same scoring scheme. We also consider each (joint) tuple to be candidate answer to the query if it contains at least one term in the query.

5.2 Efficient Exploitation & Exploration

We propose the following two algorithms to generate a weighted random sample of size k over all candidate tuples for a query.

5.2.1 Reservoir

To provide a random sample, one may calculate the total scores of all candidate answers to compute their sampling probabilities. Because this value is not known beforehand, one may use weighted reservoir sampling [7] to deliver a random sample without knowing the total score of candidate answers in a single scan of the data as follows. *Reservoir* occurs after the complete joins of the candidate network have be computed. Thus, it samples over tuples in the individual tables and tuples in the joined tables. Reservoir generates the list of answers only after computing the results of all candidate networks, therefore, users have to wait for a long time to see any result. It also computes the results of all candidate networks by performing their joins fully, which may be inefficient. We propose the following optimizations to improve its efficiency and reduce the users' waiting time.

5.2.2 Poisson-Olken

Poisson-Olken algorithm uses Poisson sampling to output progressively the selected tuples as it processes each candidate network [25]. First, when a join needs to be constructed between multiple tables in a candidate network, tuples are only joined based on some statistics collected prior to interaction. These include how likely a given tuple might join with another and how many tuples are in each relation. As tuples are joined, they are sampled immediately, allowing the algorithm to return before it has performed the entire join.

The expected value of produced tuples in the *Poisson-Olken* algorithm is close to k. However, as opposed to reservoir sampling, there is a non-zero probability that *Poisson-Olken* may deliver fewer than k tuples. To drastically reduce this chance, one may use a larger value for k in the algorithm and reject the appropriate number of the resulting tuples after the algorithm terminates [7]. The resulting algorithm will not progressively produce the sampled tuples, but, as our empirical study in Section 6 indicates, it is faster than *Reservoir* over large databases with relatively many candidate networks as it does not perform any full join. For more details, including the algorithms, see our full publication in [22].

6. EMPIRICAL STUDY

6.1 Effectiveness

It is difficult to evaluate the effectiveness of online and reinforcement learning algorithms for information systems in a live setting with real users because it requires a very long time and a large amount of resources [31, 15, 26, 14]. Thus, most studies in this area use purely simulated user interactions [26, 15]. A notable exception is [31], which uses a real-world interaction log to simulate a live interaction setting. We follow a similar approach and use Yahoo! interaction log [32] to simulate interactions using real-world queries and dataset.

Strategy Initialization: We train a user strategy over the Yahoo! 43H-interaction log whose details are in Section 3 using Roth and Erev's method, which is deemed the most accurate to model user learning according to the results of Section 3. This strategy has 341 queries and 151 intents. The DBMS starts the interaction with an empty strategy and adds queries as it receives them, initialized with equal probabilities.

Algorithms: We compare the algorithm introduced in Section 4.1 against the state-of-the-art and popular algorithm for online learning in information retrieval called UCB-1 [26, 23]. It has been shown to outperform its competitors in several studies [23, 26]. It calculates a score for an intent e given the tth submission of query q as: $Score_t(q, e) = \frac{W_{q,e,t}}{X_{q,e,t}} + \alpha\sqrt{\frac{2ln\ t}{X_{q,e,t}}}$, in which X is how many times an intent was shown to the user, W is how many times the user selects a returned intent, and α is the exploration rate set between $[0, 1]$. The first term in the formula prefers the intents that have received relatively more positive feedback, i.e., exploitation, and the second term gives higher scores to the intents that have been shown to the user less often and/or have *not* been tried for a relatively long time, i.e., exploration. UCB-1 assumes that users follow a fixed probabilistic strategy. Thus, its goal is to find the fixed but

unknown expectation of the relevance of an intent to the input query, which is roughly the first term in the formula; by minimizing the number of unsuccessful trials.

Results: We simulate the interaction of a user population that starts with our trained user strategy with UCB-1 and our algorithm. We measure the effectiveness of the algorithms using the standard metric of Reciprocal Rank (RR) [21]. In each interaction, an intent is randomly picked from the set of intents in the user strategy by its prior probability and submitted to UCB-1 and our method. Afterwards, each algorithm returns a list of 10 answers and the user clicks on the top-ranked answer that is relevant to the query according to the relevance judgment information. We run our simulations for one million interactions.

Figure 1 shows the accumulated Mean Reciprocal Rank (MRR) over all queries in the simulated interactions. Our method delivers a higher MRR than UCB-1 and its MRR keeps improving over the duration of the interaction. UCB-1, however, increases the MRR at a much slower rate. Since UCB-1 is developed for the case where users do *not* change their strategies, it learns and commits to a fixed probabilistic mapping of queries to intents quite early in the interaction. We have also observed that our method allows users to try more varieties of queries to express an intent and learn the one(s) that convey the intent effectively. As UCB-1 commits to a certain mapping of a query to an intent early in the interaction, it may *not* return sufficiently many relevant answers if the user tries this query to express another intent. This new mapping, however, could be promising in the long-run. Hence, the user and UCB-1 strategies may stabilize in less than desirable states. Since our method does *not* commit to a fixed strategy that early, users may try this query for another intent and reinforce the mapping if they get relevant answers. Thus, users have more chances to try and pick a query for an intent that will be learned and mapped effectively to the intent by the DBMS.

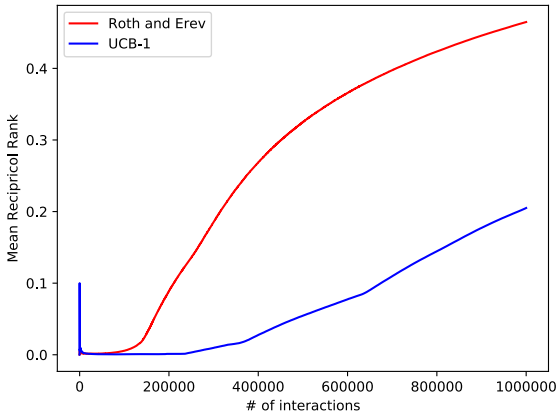

Figure 1: Mean reciprocal rank for 1,000,000 interactions

6.2 Efficiency

Databases and Queries: We have built two databases from Freebase (*developers.google.com/freebase*), *TV-Program* and *Play*. *TV-Program* contains 7 tables and consists of 291,026 tuples. *Play* contains 3 tables and consists of 8,685 tuples. For our queries, we have used two samples of 621 (459 unique) and 221 (141 unique) queries from Bing (*bing.com*) query log whose relevant answers after filtering our noisy clicks, are in *TV-program* and *Play* databases, respectively [10]. After submitting each query and getting some results, we simulate user feedback using the relevance information in the Bing log.

Query Processing: We have used Whoosh inverted index (*whoosh.readthedocs.io*) to index each table in databases. Whoosh recognizes the concept of table with multiple attributes, but cannot perform joins between different tables. Because the *Poisson-Olken* algorithm needs indexes over primary and foreign keys used to build candidate network, we have built hash indexes over these tables in Whoosh. Given an index-key, these indexes return the tuple(s) that match these keys inside Whoosh. To provide a fair comparison between *Reservoir* and *Poisson-Olken*, we have used these indexes to perform joins for both methods. We have limited the size of each candidate network to 5. Our system returns 10 tuples in each interaction for both methods.

Results: Table 6 depicts the time for processing candidate networks and reporting the results for both *Reservoir* and *Poisson-Olken* over *TV-Program* and *Play* databases over 1000 interactions. These results also show that *Poisson-Olken* is able to significantly improve the time for executing the joins in the candidate network, shown as performing joins in the table, over *Reservoir* in both databases. The improvement is more significant for the larger database, *TV-Program*. *Poisson-Olken* progressively produces tuples to show to user. But, we are not able to use this feature for all interactions. For a considerable number of interactions, *Poisson-Olken* does not produce 10 tuples, as explained in Section 5.2. Hence, we have to use a larger value of k and wait for the algorithm to finish in order to find a randomize sample of the answers as explained at the end of Section 5.2. Both methods have spent a negligible amount of time to reinforce the features, which indicate that using a rich set of features one can perform and manage reinforcement efficiently.

Table 6: Average candidate networks processing times in seconds for 1000 interactions

Database	Reservoir	Poisson-Olken
Play	0.078	0.042
TV Program	0.298	0.171

7. RELATED WORK

Database community has proposed several systems that help the DBMS learn the user's information need by showing examples to the user and collecting her feedback [19, 11, 4, 30, 2]. In these systems, a user *explicitly teaches* the system by labeling a set of examples potentially in several steps without getting any answer to her information need. Thus, the system is broken into two steps: first it learns the information need of the user by soliciting labels on certain examples from the user and then once the learning has completed, it suggests a query that may express the user's information need. These systems usually leverage active learning methods to learn the user intent by showing the fewest possible examples to the user [11]. However, ideally one would like to have a query interface in which the DBMS learns about the user's intents while answering her (vague) queries as our system does. As opposed to active learning methods, one should combine and balance exploration and learning with the normal query answering to build such a system. Moreover, current query learning systems assume that users

follow a fixed strategy for expressing their intents. Also, we focus on the problems that arise in the long-term interaction that contain more than a single query and intent.

8. CONCLUSION

Many users do *not* know how to express their information needs. We showed that users learn and modify how they express their information needs during their interaction with the DBMS and modeled the interaction between the user and the DBMS as a game, where the players would like to establish a common mapping from information needs to queries via learning. As current query interfaces do *not* effectively learn the information needs behind queries in such a setting, we proposed a reinforcement learning algorithm for the DBMS that learns the querying strategy of the user effectively. We provided efficient implementations of this learning mechanisms over large databases.

Currently the algorithm proposed in this work does not consider optimal strategy profiles. In the future we would like to have the DBMS algorithm target these optimal strategy profiles such that the mutual understanding between the two players is optimal. Another question to ask is whether there are information preserving transformations of data, that can deliver a more effective interaction, e.g. merging some entities. Given that the aforementioned transformations are costly and they do improve the interaction, we would need to find the most cost-effective ones.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1994.

[2] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *PODS*, 2013.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[4] A. Bonifati, R. Ciucanu, and S. Staworko. Learning join queries from user examples. *TODS*, 40(4), 2015.

[5] Y. Cen, L. Gan, and C. Bai. Reinforcement learning in information searching. *Information Research: An International Electronic Journal*, 18(1), 2013.

[6] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 31(3), 2006.

[7] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 263–274, New York, NY, USA, 1999. ACM.

[8] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD*, 2009.

[9] I. Cho and D. Kreps. Signaling games and stable equilibria. *Quarterly Journal of Economics*, 102, 1987.

[10] E. Demidova, X. Zhou, I. Oelze, and W. Nejdl. Evaluating Evidences for Keyword Query Disambiguation in Entity Centric Database Search. In *DEXA*, 2010.

[11] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, 2014.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[13] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In *SIGIR*, 2004.

[14] A. Grotov and M. de Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1215–1218, New York, NY, USA, 2016. ACM.

[15] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, 2013.

[16] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB 2003*.

[17] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, 2015.

[18] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, 2007.

[19] H. Li, C.-Y. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *PVLDB*, 8(13), 2015.

[20] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1262–1265, 2014.

[21] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] B. McCamish, V. Ghadakchi, A. Termehchy, B. Touri, and L. Huang. The data interaction game. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 83–98, New York, NY, USA, 2018. ACM.

[23] T. Moon, W. Chu, L. Li, Z. Zheng, and Y. Chang. An online learning framework for refining recency search results with user click feedback. *ACM Transactions on Information Systems (TOIS)*, 30(4):20, 2012.

[24] Y. Niv. The neuroscience of reinforcement learning. In *ICML*, 2009.

[25] F. Olken. *Random Sampling from Databases*. PhD thesis, University of California, Berkeley, 1993.

[26] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791. ACM, 2008.

[27] A. E. Roth and I. Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and economic behavior*, 8(1):164–212, 1995.

[28] L. Shapley. Some topics in two-person games. *Advances in game theory*, 52:1–29, 1964.

[29] H. Shteingart and Y. Loewenstein. Reinforcement learning and human behavior. *Current Opinion in Neurobiology*, 25:93–98, 04/2014 2014.

[30] Q. Tran, C. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, 2009.

[31] A. Vorobev, D. Lefortier, G. Gusev, and P. Serdyukov. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*, pages 1177–1187. International World Wide Web Conferences Steering Committee, 2015.

[32] Yahoo! Yahoo! webscope dataset anonymized Yahoo! search logs with relevance judgments version 1.0. labs.yahoo.com/Academic_Relations, 2011. [Online; accessed 5-January-2017].

[33] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *J. Comput. Syst. Sci.*, 78(5), 2012.