The Data Interaction Game

Ben McCamish
Oregon State University
mccamisb@oregonstate.edu

Vahid Ghadakchi
Oregon State University
ghadakcv@oregonstate.edu

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

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

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

ABSTRACT

As many users do *not* precisely know the structure and/or the content of databases, their queries do *not* exactly reflect their information needs. The database management systems (DBMS) may interact with users and leverage their feedback on the returned results to learn the information needs behind users' queries. Current query interfaces assume that users follow a fixed strategy of expressing their information needs, that is, the likelihood by which a user submits a query to express an information need remains unchanged during her 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. We also show that users' 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 improves the effectiveness of answering queries stochastically speaking. We analyze the challenges of efficient implementation of this method over large-scale relational databases and propose two efficient adaptations of this algorithm over large-scale relational databases. Our extensive empirical studies over real-world query workloads and large-scale relational databases indicate that our algorithms are efficient. Our empirical results also show that our proposed learning mechanism is more effective than the state-of-the-art query answering method.

KEYWORDS

collaborative interaction, game theory, database interaction, reinforcement learning

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196899

ACM Reference Format:
Ben McCamish, Vahid Ghadakchi, Arash Termehchy, Behrouz Touri, and Liang Huang. 2018. The Data Interaction Game. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196899

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 [14, 29, 30]. 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 continuing 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 understandings 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 [52], the amount of time the user spends on reading the results [23], user's eye movements [28], or the signals sent in touch-based devises [34]. 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 [10, 11]. 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 [17, 24]. 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 payoff 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 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 [15] 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 [44] in experimental game-theory.
- 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 [4]. We propose a method of answering user queries in a natural and interactive setting in Section 4 and prove that it improves the effectiveness of answering queries stochastically speaking, and converges almost surely. We show that our results hold for both the cases where the user adapts her strategy using an appropriate learning algorithm and the case where she follows a fixed strategy.
- 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 extensive 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 details. 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 [23]. 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. Next, we describe the components and structure of this interaction game for relational databases.

Basic Definitions: We fix two disjoint (countably) infinite sets of attributes and relation symbols. Every relation symbol R is associated with a set of attribute symbols denoted as $sort(R)$. Let dom be a countably infinite set of constants, e.g., strings. An instance I_R of relation symbol R with $n = |sort(R)|$ is a (finite) subset of dom^n. A *schema* S is a set of relation symbols. A database (instance) of S is a mapping over S that associates with each relation symbol R in S an instance of I_R. In this paper, we assume that dom is a set of strings.

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 [14, 30]. 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, Type, Ranking)* 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: $ans(z) \leftarrow Univ(x, \text{'}MSU\text{'}, \text{'}MI\text{'}, y, z)$.

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 [14, 30]. 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 [11, 32]. 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 $ans(t) \leftarrow Univ(x, \text{'}MSU\text{'}, y, z, t)$, 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 by which 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	Type	Rank
Missouri State University	MSU	MO	public	20
Mississippi State University	MSU	MS	public	22
Murray State University	MSU	KY	public	14
Michigan State University	MSU	MI	public	18

Table 2: Intents and Queries

2(a) Intents

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

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 [14, 26, 36]. 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 [14]. As an example consider current keyword query interfaces over relational databases [14]. Given constant v in database I and keyword w in keyword query q, let $match(v, w)$ be a function that is true if w appears in v and false otherwise. A majority of keyword query interfaces interpret keyword queries as Select-Project-Join queries that have below certain number of joins and whose *where* clauses contain only conjunctions of *match* functions [26, 36]. Using a larger subset of SQL, e.g. the ones with more joins, makes it inefficient to perform the interpretation and run its outcomes. Given schema S, the *interpretation language* of the DBMS, denoted as L, is a subset of SQL over S. We precisely define L for our implementation of DBMS strategy in Section 5. To interpret a keyword query, the DBMS searches L for the SQL queries that represent the intent behind the query as accurately as possible.

Because users may be overwhelmed by the results of many interpretations, keyword query interfaces use a deterministic real-valued scoring function to rank their interpretations and deliver only the results of top-k ones to the user [14]. It is known that such a deterministic approach may significantly limit the accuracy of interpreting queries in long-term interactions in which the information system utilizes user's feedback [3, 25, 49]. Because the DBMS shows only the result of interpretation(s) with the highest score(s) to the user, it receives feedback only on a small set of interpretations. Thus, its learning remains largely biased toward the initial set of highly ranked interpretations. For example, it may never learn that the intent behind a query is satisfied by an interpretation with a relatively low score according to the current scoring function.

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 [3, 25, 49]. Of course, the DBMS should ensure that this set of interpretations are 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 [3, 25, 49]. 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. To the best of our knowledge, to search L efficiently, current keyword query interfaces limit their search per query to a finite subset of L [14, 26, 36]. In this paper, we follow a similar approach and assume that D maps each query to only a finite subset of L. 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_2 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. The feedback signals may be noisy, e.g., a user may click on a tuple by mistake. Researchers have proposed models to accurately detect the informative signals [25]. Dealing with the issue of noisy signals is out of the scope of this paper.

The goal of both the user and the DBMS is to have as many satisfying tuples as possible in the returned tuples. Hence, both the user and the DBMS receive some payoff, i.e., reward, according to the degree by which the returned tuples match the intent. This payoff is measured based on the user feedback and using standard effectiveness metrics [37]. One example of such metrics is *precision at k*, *p@k*, which is the fraction of relevant tuples in the top-k returned tuples. At the end of each round, both the user and the DBMS receive a payoff equal to the value of the selected effectiveness metric for the returned result. We denote the payoff received by the players at each round of the game, i.e., a single interaction, for returning interpretation e_ℓ for intent e_i as $r(e_i, e_\ell)$. This payoff is computed using the user's feedback on the result of interpretation e_ℓ over the underlying database.

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.

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

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 as we will show in Section 4.

None of the players know the other player's strategy during the interaction. Given the information available to each player, it may modify its strategy at the end of each round (interaction). For example, the DBMS may reduce the probability of returning certain interpretations that has not received any positive feedback from the user in the previous rounds of the game. Let the user and DBMS strategy at round $t \in \mathbb{N}$ of the game be $U(t)$ and $D(t)$, respectively. In round $t \in \mathbb{N}$ of the game, the user and DBMS have access to the information about their past interactions. The user has access to her sequence of intents, queries, and results, the DBMS knows the sequence of queries and results, and both players have access to the sequence of payoffs (*not* expected payoffs) up to round $t-1$. It depends on the degree of rationality and abilities of the user and the DBMS how to leverage these pieces of information to improve the expected payoff of the game. For example, it may *not* be reasonable to assume that the user adopts a mechanism that requires instant access to the detailed information about her past interactions as it is not clear whether users can memorize this information for a long-term interaction. A *data interaction game* is represented as tuple $(U(t), D(t), \pi, (e^u(t-1)), (q(t-1)), (e^d(t-1)), (r(t-1)))$ in which $U(t)$ and $D(t)$ are respectively the strategies of the user and DBMS at round t, π is the prior probability of intents in U, $(e^u(t-1))$ is the sequence of intents, $(q(t-1))$ is the sequence of queries, $(e^d(t-1))$ is the sequence of interpretations, and $(r(t-1)))$ is the sequence of payoffs up to time t. Table 4 contains the notation and concept definitions introduced in this section for future reference.

3 USER LEARNING MECHANISM

It is well established that humans show reinforcement behavior in learning [40, 46]. Many lab studies with human subjects conclude

Table 4: Summary of the notations used in the model.

Notation	Definition
e_i	A user's intent
q_j	A query submitted by the user
π_i	The prior probability that the user queries for e_i
$r(e_i, e_\ell)$	The reward when the user looks for e_i and the DBMS returns e_ℓ
U	The user strategy
U_{ij}	The probability that user submits q_j for intent e_i
D	The DBMS strategy
$D_{j\ell}$	The probability that DBMS intent e_ℓ for query q_j
(U, D)	A strategy profile
$u_r(U, D)$	The expected payoff of the strategy profile (U, D) computed using reward metric r based to Equation 1

that one can model human learning using reinforcement learning models [40, 46]. 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 Reinforcement Learning Methods

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) [9, 44]. 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. The details of algorithms are in Appendix A.

3.2 Empirical Analysis

3.2.1 Interaction Logs. We use an anonymized Yahoo! interaction log for our empirical study, which consists of queries submitted to a Yahoo! search engine in July 2010 [50]. Each record in the log consists of a time stamp, user cookie id, submitted query, the top 10 results displayed to the user, and the positions of the user clicks on the returned answers. Generally speaking, typical users of Yahoo! are normal users who may not know advanced concepts, such as formal query language and schema, and use keyword queries to find their desired information. Yahoo! may generally use a combination of structured and unstructured datasets to satisfy users' intents. Nevertheless, as normal users are not aware of the existence of schema and mainly rely on the content of the returned answers to (re)formulate their queries, we expect that the users' learning mechanisms over this dataset closely resemble their learning mechanisms over structured data. We have used three different contiguous subsamples of this log whose information is shown in Table 5. The duration of each subsample is the time between the time-stamp of the first and last interaction records. Because we would like to specifically look at the users that exhibit some learning throughout their interaction, we have collected only the interactions in which a user submits at least two different queries to express the same intent. 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.

3.2.2 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 NDCG for the returned results of a query as the reward in each interaction as it models different levels of relevance [37]. The value of NDCG is between 0 and 1 and it is 1 for the most effective list.

Table 5: 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

3.2.3 Parameter Estimation. Some models, e.g., Cross's model, have some parameters that need to be trained. We have used a set of 5,000 records that appear in the interaction log immediately before the first subsample of Table 5 and found the optimal values for those parameters using grid search and the sum of squared errors.

3.2.4 Training & Testing. We train and test a single user strategy over each subsample and model, which represents the strategy of the user population in each subsample. The user strategy in each model is initialized with a uniform distribution, so that all queries are equally likely to be used for an intent. 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. We use the value of NDCG as reward for the models that use rewards to update the user strategy after each interaction. We then test the accuracy of the prediction of using a query to express an intent for each model over the remaining 10% of each subsample using the user strategy computed at the end of the training phase. Each intent is conveyed using only a single query in the testing portions of our subsamples. Hence, no learning is done in the testing phase and we do not update the user strategies. We report the mean squared errors over all intents in the testing phase for each subsample and model in Figure 1. 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.

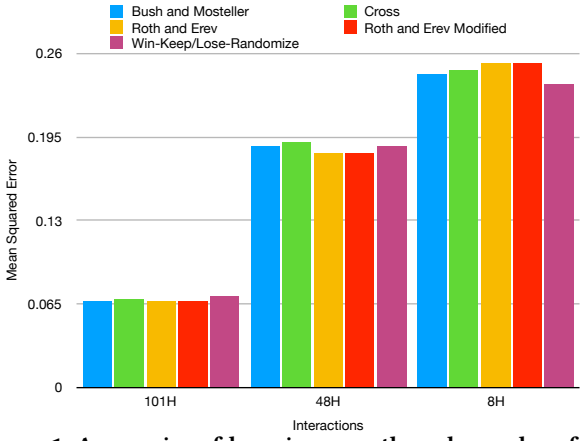

Figure 1: Accuracies of learning over the subsamples of Table 5

3.2.5 Results. Win-Keep/Lose-Randomize performs surprisingly more accurate 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 these 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.

3.2.6 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 [4]. 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 [45]. Thus, choosing the correct learning mechanism for the DBMS is crucial to improve the payoff and converge to a desired state. The following algorithmic questions are of interest:

i. How can a DBMS learn or adapt to a user's strategy?
ii. Mathematically, is a given learning algorithm effective?
iii. What would be the asymptotic behavior of a given learning algorithm?

Here, we address the first and the second questions above. Dealing with the third question is far beyond the scope and space of this paper. A summary of the notations introduced in Section 2 and used in this section can be found in Table 4.

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 $r : [m] \times [o] \rightarrow \mathbb{R}^+$ 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. One may use an available offline scoring function, e.g. [11, 26], to compute the initial reward $R(0)$ which possibly leads to an intuitive and relatively effective initial point for the learning process [49].

4.2 Analysis of the Learning Rule

We show in Section 3 that users modify their strategies in data interactions. Nevertheless, ideally, one would like to use a learning mechanism for the DBMS that accurately discovers the intents behind users' queries whether or not the users modify their strategies, as it is *not* certain that all users will always modify their strategies. Also, in some relevant applications, the user's learning is happening in a much slower time-scale compared to the learning of the DBMS. So, one can assume that the user's strategy is fixed compared to the time-scale of the DBMS adaptation. Therefore, first, we consider the case that the user is *not* adapting her strategy, i.e., she has a fixed strategy during the interaction. Then, we consider the case that the user's strategy is adapting to the DBMS's strategy but perhaps on a slower time-scale in Section 4.3.

We provide an analysis of the reinforcement mechanism provided above and will show that, statistically speaking, the adaptation rule leads to improvement of the interaction effectiveness. To simplify our analysis, we assume that the user gives feedback only on one result in the returned list of answers. Hence, we assume that the cardinality of the returned list of answers is 1. For the analysis of the learning mechanism in Section 4 and for simplification, denote

$$u(t) := u_r(U, D(t)), \tag{4}$$

for an effectiveness measure r as u_r is defined in (1).

We recall that a random process $\{X(t)\}$ is a submartingale [19] if it is absolutely integrable (i.e. $E(|X(t)|) < \infty$ for all t) and

$$E(X(t+1) \mid \mathcal{F}_t) \geq X(t),$$

where $\mathcal{F}_t$ is the history or σ-algebra generated by $X_1, \ldots, X_t$[1]. In other words, a process $\{X(t)\}$ is a sub-martingale if the expected value of $X(t+1)$ given the history $X(t), X(t-1), \ldots, X(0)$, is not strictly less than the value of $X(t)$. Note that submartingales are nothing but the stochastic counterparts of monotonically increasing sequences. As in the case of bounded (from above) monotonically increasing sequences, submartingales pose the same property, i.e. any submartingale $\{X(t)\}$ with $E(|X(t)|) < B$ for some $B \in \mathbb{R}^+$ and all $t \geq 0$ is convergent almost surely, i.e. $\lim_{t\to\infty} X(t)$ exists almost surely.

The main result in this section is that the sequence of the utilities $\{u(t)\}$ (which is indeed a stochastic process as $\{D(t)\}$ is a stochastic process) defined by (4) is a submartignale when the reinforcement learning rule in Section 4 is utilized. As a result the proposed reinforcement learning rule *stochastically* improves the efficiency of communication between the DBMS and the user. More importantly, this holds *for an arbitrary reward/effectiveness measure* r. This is rather a very strong result as the algorithm is robust to the choice of the reward mechanism.

To show this, we discuss an intermediate result. For simplicity of notation, we fix the time t and we use superscript + to denote variables at time $(t+1)$ and drop the dependencies at time t for variables depending on time t.

[1]In this case, simply we have $E(X(t+1) \mid \mathcal{F}_t) = E(X(t+1) \mid X(t), \ldots, X(1))$.

Lemma 4.1. *For any $\ell \in [m]$ and $j \in [n]$, we have*

$$E(D^+_{j\ell} \mid \mathcal{F}_t) - D_{j\ell} = D_{j\ell} \cdot \sum_{i=1}^{m} \pi_i U_{ij} \left(\frac{r_{i\ell}}{\bar{R}_j + r_{il}} - \sum_{\ell'=1}^{o} D_{j\ell'} \frac{r_{i\ell'}}{\bar{R}_j + r_{i\ell'}} \right),$$

where $\bar{R}_j = \sum_{\ell'=1}^{o} R_{j\ell'}$.

To show the main result, we use the following result in martingale theory.

Theorem 4.2. *[43] A random process $\{X(t)\}$ converges almost surely if $X(t)$ is bounded, i.e., $E(|X(t)|) < B$ for some $B \in \mathbb{R}^+$ and all $t \geq 0$ and*

$$E(X(t+1)|\mathcal{F}_t) \geq X(t) - \beta(t) \tag{5}$$

where $\beta(t) \geq 0$ is a summable sequence almost surely, i.e., $\sum_t \beta(t) < \infty$ with probability 1.

Using Lemma 4.1 and the above result, we show that up to a summable disturbance, the proposed learning mechanism is stochastically improving.

Theorem 4.3. *Let $\{u(t)\}$ be the sequence given by (4). Then,*

$$E(u(t+1 \mid \mathcal{F}_t) \geq E(u(t) \mid \mathcal{F}_t) - \beta(t),$$

for some non-negative random process $\{\beta(t)\}$ that is summable (i.e. $\sum_{t=0}^{\infty} \beta(t) < \infty$ almost surely). Hence, $\{u(t)\}$ converges almost surely.

The above result implies that the effectiveness of the DBMS, stochastically speaking, increases as time progresses when the learning rule in Section 4 is utilized. Not only that, but this property is true for cases where the feedback is not simply a 0/1 value, e.g., the selected answer may be partially relevant to the desired intent. This is indeed a desirable property for any DBMS learning scheme.

4.3 User and DBMS Adaptations

We also consider the case that the user also adapts to the DBMS's strategy. At the first glance, it may seem that if the DBMS adapts using a reasonable learning mechanism, the user's adaptation can only result in a more effective interaction as both players have identical interests. Nevertheless, it is known from the research in algorithmic game theory that in certain two-player games with identical interest in which both players adapt their strategies to improve their payoff, well-known learning methods do not converge to any (desired) stable state and cycle among several unstable states [17, 45]. Here, we focus on the identity similarity measure, i.e. we assume that $m = o$ and the user gives a boolean feedback:

$$r_{i\ell} = \begin{cases} 1 & \text{if } i = \ell, \\ 0 & \text{otherwise} \end{cases}.$$

In this case, we assume that the user adapts to the DBMS strategy at time steps $0 < t_1 < \cdots < t_k < \cdots$ and in those time-steps the DBMS is not adapting as there is no reason to assume the synchronicity between the user and the DBMS. The reinforcement learning mechanism for the user is as follows:

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

b. Let $U(0)$ be the initial user's strategy with

$$U_{ij}(0) = \frac{S_{ij}(0)}{\sum_{j'=1}^{n} S_{ij'}(0)}$$

for all $i \in [m]$ and $j \in [n]$ and let $U(t_k) = U(t_k - 1) = \cdots = U(t_{k-1} + 1)$ for all k.

c. For all $k \geq 1$, do the following:

i. The user picks a random intent $t \in [m]$ with probability π_i (independent of the earlier choices of intent) and subsequently selects a query $j \in [n]$ with probability

$$P(q(t_k) = j \mid i(t_k) = i) = U_{ij}(t_k).$$

ii. The DBMS uses the current strategy $D(t_k)$ and interpret the query by the intent $i'(t) = i'$ with probability

$$P(i'(t_k) = i' \mid q(t_k) = j) = D_{ji'}(t_k).$$

iii. User gives a reward 1 if $i = i'$ and otherwise, gives no rewards, i.e.

$$S_{ij}^{+} = \begin{cases} S_{ij}(t_k) + 1 & \text{if } j = q(t_k) \text{ and } i(t_k) = i'(t_k) \\ S_{ij}(t_k) & \text{otherwise} \end{cases}$$

where $S_{ij}^{+} = S_{ij}(t_k + 1)$.

iv. Update the user's strategy by

$$U_{ij}(t_k + 1) = \frac{S_{ij}(t_k + 1)}{\sum_{j'=1}^{n} S_{ij'}(t_k + 1)}, \tag{6}$$

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

In the above scheme $S(t)$ is the reward matrix at time t for the user.

Next, we provide an analysis of the reinforcement mechanism provided above and will show that, statistically speaking, our proposed adaptation rule for DBMS, even when the user adapts, leads to improvement of the effectiveness of the interaction. With a slight abuse of notation, let

$$u(t) := u_r(U, D(t)) = u_r(U(t), D(t)), \tag{7}$$

for an effectiveness measure r as u_r is defined in (1).

Lemma 4.4. *Let $t = t_k$ for some $k \in \mathbb{N}$. Then, for any $i \in [m]$ and $j \in [n]$, we have*

$$E(U_{ij}^{+} \mid \mathcal{F}_t) - U_{ij} = \frac{\pi_i U_{ij}}{\sum_{\ell=1}^{n} S_{i\ell} + 1}(D_{ji} - u^i(t)) \tag{8}$$

where

$$u^i(t) = \sum_{j=1}^{n} U_{ij}(t) D_{ji}(t).$$

Using Lemma 4.4, we show that the process $\{u(t)\}$ is a sub-martingale.

Theorem 4.5. *Let $t = t_k$ for some $k \in \mathbb{N}$. Then, we have*

$$E(u(t + 1) \mid \mathcal{F}_t) - u(t) \geq 0 \tag{9}$$

where $u(t)$ is given by (7).

Corollary 4.6. *The sequence $\{u(t)\}$ given by* (4) *converges almost surely.*

The authors in [27] have also analyzed the effectiveness of a 2-player signaling game in which both players use Roth and Erev's model for learning. However, they assume that both players learn at the same time-scale. Our result in this section holds for the case where users and DBMS learn at different time-scales, which may arguably be the dominant case in our setting as generally users may learn in a much slower time-scale compared to the DBMS.

5 EFFICIENT QUERY ANSWERING OVER RELATIONAL DATABASES

An efficient implementation of the 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 [14, 26]. Due to a series of seminal works by database researchers [22], there are efficient algorithms to find such a list of answers. Nevertheless, our reinforcement learning algorithm uses a randomized semantic 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 [12, 29]. 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 Maintaining DBMS Strategy

5.1.1 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 [14]. A notable example of such systems is IR-Style [26]. As it is mentioned in Section 2.4, given a keyword query, these systems translate the input query to a Select-Project-Join query whose *where* clause contains function *match*. The results of these interpretations are computed, scored according to some ranking function, and are returned to the user. We provide an overview of the basic concepts of such a system. We refer the reader to [14, 26] 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. The query interface may also use a scoring function, e.g., traditional TF-IDF text matching score, to measure how exactly each tuple in a tuple-set matches some terms in q.

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 leafs are tuple-sets. For instance, one candidate network for the aforementioned database of products is *Product* ⋈ *ProductCustomer* ⋈ *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. For each candidate network, the query interface runs a SQL query and return its results to the users.There are algorithms to reduce the running time of this stage, e.g., run only the SQL queries guaranteed to produce top-k tuples [26]. 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. 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.

5.1.2 Managing Reinforcements. The aforementioned keyword query interface implements a basic DBMS strategy of mapping queries to results but it does not leverage users' feedback and adopts a deterministic strategy without any exploration. A naive way to record users' reinforcement is to maintain a mapping from queries to tuples and directly record the reinforcements applied to each pair of query and tuple. In this method, the DBMS has to maintain the list of all submitted queries and returned tuples. Because many returned tuples are the joint tuples produced by candidate networks, it will take an enormous amount of space and is inefficient to update. Hence, instead of recording reinforcements directly for each pair of query and tuple, we construct some features for queries and tuples and maintain the reinforcement in the constructed feature space. More precisely, we construct and maintain a set of *n-gram* features for each attribute value in the base relations and each input query. N-grams are contiguous sequences of terms in a text and are widely used in text analytics and retrieval [37]. In our implementation, we use up to 3-gram features to model the challenges in managing a large set of features. Each feature in every attribute value in the database has its associated attribute and relation names to reflect the structure of the data. We maintain a reinforcement mapping from query features to tuple features. After a tuple gets reinforced by the user for an input query, our system increases the reinforcement value for the Cartesian product of the features in the query and the ones in the reinforced tuple. According to our experiments in Section 6, this reinforcement mapping can be efficiently maintained in the main memory by only a modest space overhead.

Given an input query q, our system computes the score of each tuple t in every tuple-set using the reinforcement mapping: it finds the n-gram features in t and q and sums up their reinforcement values recorded in the reinforcement mapping. Our system may use a weighted combination of this reinforcement score and traditional text matching score, e.g., TF-IDF score, to compute the final score. One may also weight each tuple feature proportional to its inverse frequency in the database similar to some traditional relevance feedback models [37]. Due to the space limit, we mainly focus on developing an efficient implementation of query answering based on reinforcement learning over relational databases and leave using more advanced scoring methods for future work. The scores of joint tuples are computed as it is explained in Section 5.1.1. We will explain in Section 5.2, how we convert these scores to probabilities and return tuples. Using features to compute and record user feedback has also the advantage of using the reinforcement of a pair of query and tuple to compute the relevance score of other tuples for other queries that share some features. Hence, reinforcement for one query can be used to return more relevant answers to other queries.

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 [13] to deliver a random sample without knowing the total score of candidate answers in a single scan of the data as follows.

Algorithm 1 Reservoir

```
W ← 0
Initialize reservoir array A[k]to kdummy tuples.
for all candidate network CN do
    for all t ∈ CN do
        if A has dummy values then
            insert k copies of t into A
        else
            W ← W + Sc(t)
            for all i = 1 ∈ k do
                insert t into A[i] with probability Sc(t)/W
```

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. It selects the tuple t with probability $\frac{Sc(t)}{M}$, where M is an upper bound to the total scores of all candidate answers. To compute M, we use the following heuristic. Given candidate network CN, we get the upper bound for the total score of all tuples generated from CN: $M_{CN} = \frac{1}{n}(\sum_{TS \in CN} Sc_{max}(TS))$ $\frac{1}{2}\Pi_{TS \in CN}|TS|$ in which $Sc_{max}(TS)$ is the maximum query score of tuples in the tuple-set TS and $|TS|$ is the size of each tuple-set. The term $\frac{1}{n}(\sum_{TS \in CN} Sc_{max}(TS))$ is an upper bound to the scores of tuples generated by CN. Since each tuple generated by CN must

contain one tuple from each tuple-set in CN, the maximum number of tuples in CN is $\Pi_{TS \in CN}|TS|$. It is very unlikely that all tuples of every tuple-set join with all tuples in every other tuple-set in a candidate network. Hence, we divide this value by 2 to get a more realistic estimation. We do not consider candidate networks with cyclic joins, thus, each tuple-set appears at most once in a candidate network. The value of M is the sum of the aforementioned values for all candidate networks with size greater than one and the total scores of tuples in each tuple-set. Since the scores of tuples in each tuple-set is kept in the main memory, the maximum and total scores and the size of each tuple-set is computed efficiently before computing the results of any candidate network.

Both *Reservoir* and the aforementioned Poisson sampling compute the full joins of each candidate network and then sample the output. This may take a long time particularly for candidate networks with some base relations. There are several join sampling methods that compute a sample of a join by joining only samples the input tables and avoid computing the full join [13, 31, 41]. To sample the results of join $R_1 \bowtie R_2$, most of these methods must know some statistics, such as the number of tuples in R_2 that join with each tuple in R_1, before performing the join. They precompute these statistics in a preprocessing step for each base relation. But, since R_1 and/or R_2 in our candidate networks may be tuples sets, one cannot know the aforementioned statistics unless one performs the full join.

However, the join sampling algorithm proposed by Olken [41] finds a random sample of the join without the need to precompute these statistics. Given join $R_1 \bowtie R_2$, let $t \rtimes R_2$ denote the set of tuples in R_2 that join with $t \in R_1$, i.e., the right semi-join of t and R_2. Also, let $|t \rtimes R_2|_{max}^{t \in R_1}$ be the maximum number of tuples in R_2 that join with a single tuple $t \in R_1$. The Olken algorithm first randomly picks a tuple t_1 from R_1. It then randomly selects the tuple t_2 from $t_1 \rtimes R_2$. It accepts the joint tuple $t_1 \bowtie t_2$ with probability $\frac{|t_1 \rtimes R_2|}{|t \rtimes R_2|_{max}^{t \in R_1}}$ and rejects it with the remaining probability. To avoid scanning R_2 multiple times, Olken algorithm needs an index over R_2. Since the joins in our candidate networks are over only primary and foreign keys, we do not need too many indexes to implement this approach.

We extend the Olken algorithm to sample the results of a candidate network without doing its joins fully as follows. Given candidate network $R_1 \bowtie R_2$, our algorithm randomly samples tuple $t_1 \in R_1$ with probability $\frac{Sc(t_1)}{\sum_{t \in R_1}(Sc(t))}$, where $Sc(t)$ is the score of tuple t, if R_1 is a tuple-set. Otherwise, if R_1 is a base relation, it picks the tuple with probability $\frac{1}{|R_1|}$. The value of $\sum_{t \in R}(Sc(t))$ for each tuple set R is computed at the beginning of the query processing and the value of $|R|$ for each base relation is calculated in a preprocessing step. The algorithm then samples tuple t_2 from $t_1 \rtimes R_2$ with probability $\frac{Sc(t_2)}{\sum_{t \in t_1 \rtimes R_2}(Sc(t))}$ if R_2 is a tuple-set and $\frac{1}{|t_1 \rtimes R_2|}$ if R_2 is a base relation. It accepts the joint tuple with probability $\frac{\sum_{t \in t_1 \rtimes R_2} Sc(t)}{\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t))}$ and rejects it with the remaining probability.

To compute the exact value of $\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t))$, one has to perform the full join of R_1 and R_2. Hence, we use an upper bound on $\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t))$ in Olken algorithm. Using an upper bound for this value, Olken algorithm produces a correct random sample but it may reject a larger number of tuples and generate a smaller number of samples. To compute an upper bound on the value of $\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t))$, we precompute the value of $|t \rtimes B_i|_{max}^{t \in B_j}$ before the query time for all base relations B_i and B_j with primary and foreign keys of the same domain of values. Assume that B_1 and B_2 are the base relations of tuple-sets R_1 and R_2, respectively. We have $|t \rtimes R_2|_{max}^{t \in R_1} \leq |t \rtimes B_2|_{max}^{t \in B_1}$. Because $\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t)) \leq \max_{t \in R_2}(Sc(t))|t \rtimes R_2|_{max}^{t \in R_1}$, we have $\max(\sum_{t \in s \rtimes R_2, s \in R_1} Sc(t)) \leq \max_{t \in R_2}(Sc(t))|t \rtimes B_2|_{max}^{t \in B_1}$. Hence, we use $\frac{\sum_{t \in t_1 \rtimes R_2} Sc(t)}{\max_{t \in R_2}(Sc(t))|t \rtimes B_2|_{max}^{t \in B_1}}$ for the probability of acceptance. We iteratively apply the aforementioned algorithm to candidate networks with multiple joins by treating the join of each two relations as the first relation for the subsequent join in the network.

The following algorithm adopts a Poisson sampling method to return a sample of size k over all candidate networks using the aforementioned join sampling algorithm. We show binomial distribution with parameters n and p as $B(n, p)$. We denote the aforementioned join algorithm as *Extended-Olken*. Also, *ApproxTotalScore* denotes the approximated value of total score computed as explained at the beginning of this section.

Algorithm 2 Poisson-Olken

```
x ← k
W ← ApproxTotalScore / k
while x > 0 do
    for all candidate network CN do
        if CN is a single tuple-set then
            for all t ∈ CN do
                output t with probability Sc(t)/W
                if a tuple t is picked then
                    x ← x − 1
        else
            let CN = R_1 ⋈ ... ⋈ R_n
            for all t ∈ R_1 do
                Pick value X from distribution B(k, Sc(t)/W)
                Pipeline X copies of t to the Olken algorithm
                if Olken accepts m tuples then
                    x ← x − m
```

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 [13]. 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.

6 EMPIRICAL STUDY

6.1 Effectiveness

6.1.1 Experimental Setup. 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 [24, 25, 42, 47, 49]. Thus, most studies in this area use purely simulated user interactions [25, 42, 47]. A notable expectation is [49], which uses a real-world interaction log to simulate a live interaction setting. We follow a similar approach and use Yahoo! interaction log [50] to simulate interactions using real-world queries and dataset.

User 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 Yahoo! interaction log contains user clicks on the returned intents, i.e. URLs. However, a user may click a URL by mistake [49]. We consider only the clicks that are not noisy according to the relevance judgment information that accompanies the interaction log. According to the empirical study reported in Section 3.2, the parameters of number and length of sessions and the amount of time between consecutive sessions do *not* impact the user learning mechanism in long-term communications. Thus, we have *not* organized the generated interactions into sessions.

Metric: Since almost all returned results have only one relevant answer and the relevant answers to all queries have the same level of relevance, we measure the effectiveness of the algorithms using the standard metric of Reciprocal Rank (RR) [37]. RR is $\frac{1}{r}$ where r is the position of the first relevant answer to the query in the list of the returned answers. RR is particularly useful where each query in the workload has a very few relevant answers in the returned results, which is the case for the queries used in our experiment.

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 [3, 39, 42, 49]. It has been shown to outperform its competitors in several studies [39, 42]. 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.

Parameter Estimation: We randomly select 50% of the intents in the trained user strategy to learn the exploration parameter α in UCB-1 using grid search and sum of squared errors over 10,000 interactions that are after the interactions in the 43H-interaction log. We do *not* use these intents to compare algorithms in our simulation. We calculate the prior probabilities, π in Equation 1, for the intents in the trained user strategy that are *not* used to find the parameter of UCB-1 using the entire Yahoo! interaction log.

DBMS Strategy Initialization: The DBMS starts the interaction with an strategy that does *not* have any query. Thus, the DBMS is *not* aware of the set of submitted queries apriori. When the DBMS sees a query for the first time, it stores the query in its strategy, assigns equal probabilities for all intents to be returned for this query, returns some intent(s) to answer the query, and stores the user feedback on the returned intent(s) in the DBMS strategy. If the DBMS has already encountered the query, it leverages the previous user's feedback on the results of this query and returns the set of intents for this query using our proposed learning algorithm. Retrieval systems that leverage online learning perform some filtering over the initial set of answers to make efficient and effective exploration possible [25, 49]. More precisely, to reduce the set of alternatives over a large dataset, online and reinforcement learning algorithms apply a traditional selection algorithm to reduce the number of possible intents to a manageable size. Otherwise, the learning algorithm has to explore and solicit user feedback on numerous items, which takes a very long time. For instance, online learning algorithms used in searching a set of documents, e.g., UCB-1, use traditional information retrieval algorithms to filter out obviously non-relevant answers to the input query, e.g., the documents with low TF-IDF scores. Then, they apply the exploitation-exploration paradigm and solicit user feedback on the remaining candidate answers. The Yahoo! interaction workload has all queries and intents anonymized, thus we are unable to perform a filtering method of our own choosing. Hence, we use the entire collection of possible intents in the portion of the Yahoo! query log used for our simulation. This way, there 4521 intent per query that can be returned, which is close to the number of answers a reinforcement learning algorithm may consider over a large data set after filtering [49]. The DBMS strategy for our method is initialized to be completely random.

6.1.2 Results. We simulate the interaction of a user population that starts with our trained user strategy with UCB-1 and our algorithm. 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. The details of simulation is reported in our technical report [38]. We run our simulations for one million interactions.

Figure 2 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. Hence, it cannot learn as effectively as our algorithm where users modify their strategies using a randomized method, such as Roth and Erev's. As our method is more exploratory than UCB-1, it enables users to provide feedback on more varieties of intents than they do for UCB-1. This enables our method to learn more accurately how users express their intents in the long-run.

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. We have discussed and proposed solutions for mitigating the startup period of our algorithm in Appendix E.

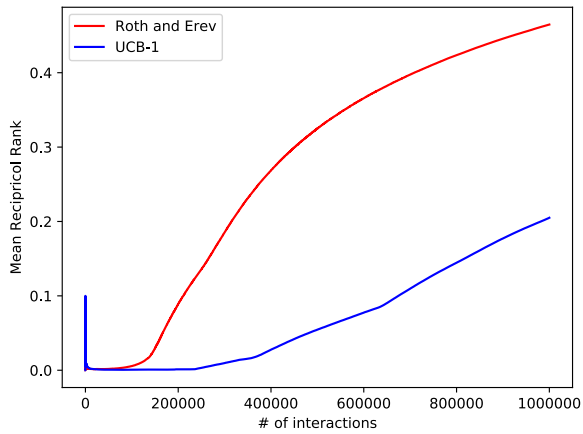

Figure 2: Mean reciprocal rank for 1,000,000 interactions

6.2 Efficiency

6.2.1 Experimental Setup. **Databases and Queries:** We have built two databases from Freebase (*developers.google.com/freebase*), *TV-Program* and *Play*. *TV-Program* contains 7 tables and consisting of 291,026 tuples. *Play* contains 3 tables and consisting 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 [20]. 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 build 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 join for both methods. We also precompute and maintain all 3-grams of the tuples in each database as mentioned in Section 5.1. We have implemented our system using both *Reservoir* and *Poisson* algorithms. We have limited the size of each candidate network to 5. Our system returns 10 tuples in each interaction for both methods.

Hardware Platform: We run experiments on a server with 32 2.6GHz Intel Xeon E5-2640 processors with 50GB of main memory.

6.2.2 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

Query learning: 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 [2, 7, 18, 33, 48]. 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 [18]. 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. A review of other related works is in the appendix C.

8 CONCLUSION

Many users do *not* know how to express their information needs. A DBMS may interact with these users and learn 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.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1994. *Foundations of Databases: The Logical Level.* Addison-Wesley.

[2] Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. 2013. Learning and verifying quantified boolean queries by example. In *PODS*.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

[4] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.

[5] Paolo Avesani and Marco Cova. 2005. Shared lexicon for distributed annotations on the Web. In *WWW*.

[6] J. A. Barrett and K. Zollman. 2008. The Role of Forgetting in the Evolution and Learning of Language. *Journal of Experimental and Theoretical Artificial Intelligence* 21, 4 (2008), 293–309.

[7] Angela Bonifati, Radu Ciucanu, and Slawomir Staworko. 2015. Learning Join Queries from User Examples. *TODS* 40, 4 (2015).

[8] Robert R Bush and Frederick Mosteller. 1953. A stochastic model with applications to learning. *The Annals of Mathematical Statistics* (1953), 559–585.

[9] Yonghua Cen, Liren Gan, and Chen Bai. 2013. Reinforcement Learning in Information Searching. *Information Research: An International Electronic Journal* 18, 1 (2013).

[10] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. 2009. Query Recommendations for Interactive Database Exploration. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM 2009)*. Springer-Verlag, Berlin, Heidelberg, 3–18. https://doi.org/10.1007/978-3-642-02279-1_2

[11] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. 2006. Probabilistic Information Retrieval Approach for Ranking of Database Query Results. *TODS* 31, 3 (2006).

[12] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. 511–519. https://doi.org/10.1145/3035918.3056097

[13] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On Random Sampling over Joins. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*. ACM, New York, NY, USA, 263–274. https://doi.org/10.1145/304182.304206

[14] Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. 2009. Keyword Search on Structured and Semi-structured Data. In *SIGMOD*.

[15] I. Cho and D. Kreps. 1987. Signaling games and stable equilibria. *Quarterly Journal of Economics* 102 (1987).

[16] John G Cross. 1973. A stochastic learning model of economic behavior. *The Quarterly Journal of Economics* 87, 2 (1973), 239–266.

[17] Constantinos Daskalakis, Rafael Frongillo, Christos H. Papadimitriou, George Pierrakos, and Gregory Valiant. 2010. On Learning Algorithms for Nash Equilibria. In *Proceedings of the Third International Conference on Algorithmic Game Theory (SAGT'10)*. Springer-Verlag, Berlin, Heidelberg, 114–125. http://dl.acm.org/citation.cfm?id=1929237.1929248

[18] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. 2014. Explore-by-example: An Automatic Query Steering Framework for Interactive Data Exploration. In *SIGMOD*.

[19] Rick Durrett. 2010. *Probability: theory and examples*. Cambridge university press.

[20] Elena Demidova and Xuan Zhou and Irina Oelze and Wolfgang Nejdl. 2010. Evaluating Evidences for Keyword Query Disambiguation in Entity Centric Database Search. In *DEXA*.

[21] Ido Erev and Alvin E Roth. 1995. *On the Need for Low Rationality, Gognitive Game Theory: Reinforcement Learning in Experimental Games with Unique, Mixed Strategy Equilibria.*

[22] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01)*. ACM, New York, NY, USA, 102–113. https://doi.org/10.1145/375551.375567

[23] Laura A. Granka, Thorsten Joachims, and Geri Gay. 2004. Eye-tracking Analysis of User Behavior in WWW Search. In *SIGIR*.

[24] Artem Grotov and Maarten de Rijke. 2016. 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)*. ACM, New York, NY, USA, 1215–1218. https://doi.org/10.1145/2911451.2914798

[25] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2013. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval* 16, 1 (2013), 63–90.

[26] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. [n. d.]. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB 2003*.

[27] Yilei Hu, Brian Skyrms, and Pierre Tarrès. 2011. Reinforcement learning in signaling game. *arXiv preprint arXiv:1103.5818* (2011).

[28] Jeff Huang, Ryen White, and Georg Buscher. 2012. User See, User Point: Gaze and Cursor Alignment in Web Search. In *CHI*.

[29] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. In *SIGMOD*.

[30] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. 2007. Making Database Systems Usable. In *SIGMOD*.

[31] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *SIGMOD*. 631–646. https://doi.org/10.1145/2882903.2882940

[32] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. 2010. SnipSuggest: Context-aware Autocompletion for SQL. *PVLDB* 4, 1 (2010).

[33] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query From Examples: An Iterative, Data-Driven Approach to Query Construction. *PVLDB* 8, 13 (2015).

[34] Erietta Liarou and Stratos Idreos. 2014. 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*. 1262–1265. https://doi.org/10.1109/ICDE.2014.6816756

[35] Jiyun Luo, Sicong Zhang, and Hui Yang. 2014. Win-Win Search: Dual-Agent Stochastic Game in Session Search. In *SIGIR*.

[36] Yi Luo, Xumein Lin, Wei Wang, and Xiaofang Zhou. [n. d.]. SPARK: Top-k Keyword Query in Relational Databases. In *SIGMOD 2007*.

[37] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *An Introduction to Information Retrieval*. Cambridge University Press.

[38] Ben McCamish, Arash Termehchy, and Behrouz Touri. 2016. A Signaling Game Approach to Databases Querying and Interaction. *arXiv preprint arXiv:1603.04068* (2016).

[39] Taesup Moon, Wei Chu, Lihong Li, Zhaohui Zheng, and Yi Chang. 2012. An online learning framework for refining recency search results with user click feedback. *ACM Transactions on Information Systems (TOIS)* 30, 4 (2012), 20.

[40] Yael Niv. 2009. The Neuroscience of Reinforcement Learning. In *ICML*.

[41] Frank Olken. 1993. *Random Sampling from Databases*. Ph.D. Dissertation. University of California, Berkeley.

[42] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. 2008. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*. ACM, 784–791.

[43] Herbert Robbins and David Siegmund. 1985. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*. Springer.

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

[45] Lloyd S Shapley et al. 1964. Some topics in two-person games. *Advances in game theory* 52, 1-29 (1964), 1–2.

[46] Hanan Shteingart and Yonatan Loewenstein. 2014. Reinforcement learning and human behavior. *Current Opinion in Neurobiology* 25 (04/2014 2014), 93–98.

[47] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. 2013. Ranked bandits in metric spaces: learning diverse rankings over large document collections. *Journal of Machine Learning Research* 14, Feb (2013), 399–436.

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

[49] Aleksandr Vorobev, Damien Lefortier, Gleb Gusev, and Pavel Serdyukov. 2015. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*. International World Wide Web Conferences Steering Committee, 1177–1187.

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

[51] Zhepeng Yan, Nan Zheng, Zachary G Ives, Partha Pratim Talukdar, and Cong Yu. 2013. Actively soliciting feedback for query answers in keyword search-based data integration. In *Proceedings of the VLDB Endowment*, Vol. 6. VLDB Endowment, 205–216.

[52] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. The K-armed Dueling Bandits Problem. *J. Comput. Syst. Sci.* 78, 5 (2012).

[53] Yinan Zhang and ChengXiang Zhai. 2015. Information Retrieval as Card Playing: A Formal Model for Optimizing Interactive Retrieval Interface. In *SIGIR*.

A USER LEARNING METHODS

Win-Keep/Lose-Randomize This method uses only the most recent interaction for an intent to determine the queries used to express the intent in the future [6]. Assume that the user conveys an intent e by a query q. If the reward of using q is above a specified threshold τ the user will use q to express e in the future. Otherwise, the user randomly picks another query uniformly at random to express e.

Bush and Mosteller's Model: Bush and Mosteller's model increases the probability that a user will choose a given query to express an intent by an amount proportional to the reward of using that query and the current probability of using this query for the intent [8]. If a user receives reward r for using $q(t)$ at time t to express intent e_i, the model updates the probabilities of using queries in the user strategy as follows.

$$U_{ij}(t+1) = \begin{cases} U_{ij}(t) + \alpha^{BM} \cdot (1 - U_{ij}(t)) & q_j = q(t) \wedge r \geq 0 \\ U_{ij}(t) - \beta^{BM} \cdot U_{ij}(t) & q_j = q(t) \wedge r < 0 \end{cases} \tag{10}$$

$$U_{ij}(t+1) = \begin{cases} U_{ij}(t) - \alpha^{BM} \cdot U_{ij}(t) & q_j \neq q(t) \wedge r \geq 0 \\ U_{ij}(t) + \beta^{BM} \cdot (1 - U_{ij}(t)) & q_j \neq q(t) \wedge r < 0 \end{cases} \tag{11}$$

$\alpha^{BM} \in [0,1]$ and $\beta^{BM} \in [0,1]$ are parameters of the model. Since effectiveness metrics in interaction are always greater than zero, β^{BM} is never used in our experiments.

Cross's Model: Cross's model modifies the user's strategy similar to Bush and Mosteller's model [16], but uses the amount of the received reward to update the user strategy. Given a user receives reward r for using $q(t)$ at time t to express intent e_i, we have:

$$U_{ij}(t+1) = \begin{cases} U_{ij}(t) + R(r) \cdot (1 - U_{ij}(t)) & q_j = q(t) \\ U_{ij}(t) - R(r) \cdot U_{ij}(t) & q_j \neq q(t) \end{cases} \tag{12}$$

$$R(r) = \alpha^C \cdot r + \beta^C \tag{13}$$

Parameters $\alpha^C \in [0,1]$ and $\beta^C \in [0,1]$ are used to compute the adjusted reward $R(r)$ based on the value of actual reward r.

Roth and Erev's Model: Roth and Erev's model reinforces the probabilities directly from the reward value r that is received when the user uses query $q(t)$ [44]. Its most important difference with other models is that it explicitly accumulates all the rewards gained by using a query to express an intent. $S_{ij}(t)$ in matrix $S(t)$ maintains the accumulated reward of using query q_j to express intent e_i over the course of interaction up to round (time) t.

$$S_{ij}(t+1) = \begin{cases} S_{ij}(t) + r & q_j = q(t) \\ S_{ij}(t) & q_j \neq q(t) \end{cases} \tag{14}$$

$$U_{ij}(t+1) = \frac{S_{ij}(t+1)}{\sum_{j'}^{n} S_{ij'}(t+1)} \tag{15}$$

Each query *not* used in a successful interaction will be implicitly penalized as when the probability of a query increases, all others will decrease to keep U row-stochastic.

Roth and Erev's Modified Model: Roth and Erev's modified model is similar to the original Roth and Erev's model, but it has an additional parameter that determines to what extent the user takes in to account the outcomes of her past interactions with the system [21]. It is reasonable to assume that the user may forget the results of her much earlier interactions with the system. This is accounted for by the *forget* parameter $\sigma \in [0,1]$. Matrix $S(t)$ has the same role it has for the Roth and Erev's model.

$$S_{ij}(t+1) = (1 - \sigma) \cdot S_{ij}(t) + E(j, R(r)) \tag{16}$$

$$E(j, R(r)) = \begin{cases} R(r) \cdot (1 - \epsilon) & q_j = q(t) \\ R(r) \cdot (\epsilon) & q_j \neq q(t) \end{cases} \tag{17}$$

$$R(r) = r - r_{min} \tag{18}$$

$$U_{ij}(t+1) = \frac{S_{ij}(t+1)}{\sum_{j'}^{n} S_{ij'}(t+1)} \tag{19}$$

In the aforementioned formulas, $\epsilon \in [0,1]$ is a parameter that weights the reward that the user receives, n is the maximum number of possible queries for a given intent e_i, and r_{min} is the minimum expected reward that the user wants to receive. The intuition behind this parameter is that the user often assumes some minimum amount of reward is guaranteed when she queries the database. The model uses this minimum amount to discount the received reward. We set r_{min} to 0 in our analysis, representing that there is no expected reward in an interaction.

Latest-Reward: The Latest-Reward method reinforces the user strategy based on the previous reward that the user has seen when querying for an intent e_i. All other queries have an equal probability to be chosen for a given intent. Let a user receive reward $r \in [0,1]$ by entering query q_j to express intent e_i. The Latest-Reward method sets the probability of using q_j to convey e_i in the user strategy, U_{ij}, to r and distribute the remaining probability mass $1 - r$ evenly between other entries related to intent e_i, in U_{ik}, where $k \neq j$.

B MISSING PROOFS

Proof of Lemma 4.1: Fix $\ell \in [m]$ and $j \in [n]$. Let A be the event that at the t'th iteration, we reinforce a pair (j, ℓ') for some $\ell' \in [m]$. Then on the complement A^c of A, $D^+_{j\ell}(\omega) = D_{j\ell}(\omega)$. Let $A_{i,\ell'} \subseteq A$ be the subset of A such that the intent of the user is i and the pair (j, ℓ') is reinforced. Note that the collection of sets $\{A_{i,\ell'}\}$ for $i, \ell' \in [m]$, are pairwise mutually exclusive and their union constitute the set A.

We note that

$$D^+_{j\ell} = \sum_{i=1}^{m} \left(\frac{R_{j\ell} + r_{il}}{\bar{R}_j + r_{i\ell}} 1_{A_{i,\ell}} + \sum_{\substack{\ell'=1 \\ \ell' \neq \ell}}^{o} \frac{R_{j\ell}}{\bar{R}_j + r_{i\ell'}} 1_{A_{i,\ell'}} \right) + D_{j\ell} 1_{A^c}.$$

Therefore, we have

$$E(D^+_{j\ell} \mid \mathcal{F}_t) = \sum_{i=1}^{m} \pi_i U_{ij} D_{j\ell} \frac{R_{j\ell} + r_{i\ell}}{\bar{R}_j + r_{i\ell}} + \sum_{i=1}^{m} \pi_i U_{ij} \sum_{\ell \neq \ell'} D_{j\ell'} \frac{R_{j\ell}}{\bar{R}_j + r_{i\ell'}} + (1-p) D_{j\ell},$$

where $p = \mathbb{P}(A \mid \mathcal{F})$. Note that $D_{j\ell} = \frac{R_{ji}}{\bar{R}_j}$ and hence,

$$E(D^+_{j\ell} \mid \mathcal{F}_t) - D_{j\ell} = \sum_{i=1}^{m} \pi_i U_{ij} D_{j\ell} \frac{r_{i\ell} \bar{R}_j - R_{j\ell}}{\bar{R}_j (\bar{R}_j + r_{i\ell})} - \sum_{i=1}^{m} \pi_i U_{ij} \sum_{\ell \neq \ell'} D_{j\ell'} \frac{R_{j\ell} r_{i\ell'}}{\bar{R}_j (\bar{R}_j + r_{i\ell'})}.$$

Replacing $\frac{R_{jl}}{\bar{R}_j}$ with $D_{j\ell}$ and rearranging the terms in the above expression, we get the result. □

Proof of Theorem 4.3: Let $u^+ := u(t+1)$, $u := u(t)$,

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

and also define $\bar{R}_j := \sum_{\ell'=1}^{m} R_{j\ell'}$. Note that u^j is the efficiency of the jth signal/query.

Using the linearity of conditional expectation and Lemma 4.1, we have:

$$\begin{aligned} E(u^+ \mid \mathcal{F}_t) - u &= \sum_{i=1}^{m}\sum_{j=1}^{n} \pi_i U_{ij} \sum_{\ell=1}^{o} r_{i\ell'} \left(E(D^+_{j\ell} \mid \mathcal{F}_t) - D_{j\ell} \right) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{\ell=1}^{o} \pi_i U_{ij} D_{j\ell} r_{i\ell} \left(\sum_{i'=1}^{m} \pi'_i U_{i'j} \left(\frac{r_{i'\ell}}{\bar{R}_j + r_{i'\ell}} \right.\right. \\ &\qquad \left.\left. - \sum_{\ell'=1}^{o} D_{j\ell'} \frac{r_{i'\ell'}}{\bar{R}_j + r_{i'\ell'}} \right) \right). \end{aligned} \tag{20}$$

Now, let $y_{j\ell} = \sum_{i=1}^{m} \pi_i U_{ij} r_{i\ell}$ and $z_{j\ell} = \sum_{i=1}^{m} \pi_i U_{ij} \frac{r_{i\ell}}{\bar{R}_j + r_{i\ell}}$. Then, we get from the above expression that

$$\begin{aligned} &E(u^+ \mid \mathcal{F}_t) - u = \\ &\qquad \sum_{j=1}^{n} \left(\sum_{\ell=1}^{o} D_{j\ell} y_{i\ell} z_{j\ell} - \sum_{\ell=1}^{o} D_{j\ell} y_{j\ell} \sum_{\ell'=1}^{o} D_{j\ell'} z_{j\ell'} \right). \end{aligned} \tag{21}$$

Now, we express the above expression as

$$E(u^+ \mid \mathcal{F}_t) - u = V_t + \tilde{V}_t \tag{22}$$

where

$$V_t = \sum_{j=1}^{n} \frac{1}{\bar{R}_j} \left(\sum_{\ell=1}^{o} D_{j\ell} y^2_{j\ell} - \left(\sum_{l=1}^{o} D_{j\ell} y_{j\ell} \right)^2 \right),$$

and

$$\tilde{V}_t = \sum_{j=1}^{n} \left(\sum_{\ell=1}^{o} D_{j\ell} y_{j\ell} \sum_{\ell'=1}^{o} D_{j\ell'} \tilde{z}_{j\ell'} - \sum_{\ell=1}^{m} D_{j\ell} y_{j\ell} \tilde{z}_{j\ell} \right). \tag{23}$$

Further, $\tilde{z}_{j\ell} = \sum_{i=1} \pi_i U_{ij} \frac{r^2_{i\ell}}{\bar{R}_j(\bar{R}_j + r_{i\ell})}$.

We claim that $V_t \geq 0$ for each t and $\{\tilde{V}_t\}$ is a summable sequence almost surely. Then, from (22) and Theorem 4.2, we get that $\{u_t\}$ converges almost surely and it completes the proof. Next, we validate our claims.

We first show that $V_t \geq 0, \forall t$. Note that D is a row-stochastic matrix and hence, $\sum_{\ell=1}^{o} D_{j\ell} = 1$. Therefore, by the Jensen's inequality [19], we have:

$$\sum_{\ell=1}^{o} D_{j\ell} (y_{j\ell})^2 \geq \sum_{\ell=1}^{o} (D_{j\ell} y_{j\ell})^2.$$

Hence, $V \geq 0$.

We next claim that $\{\tilde{V}_t\}$ is a summable sequence with probability one. It can be observed from (23) that

$$V_t \leq \sum_{j=1}^{o} \frac{o^2 n}{\bar{R}^2_j}. \tag{24}$$

since $y_{j\ell} \leq 1, \tilde{z}_{j\ell} \leq \bar{R}_j^{-2}$ for each $j \in [n], \ell \in [m]$ and D is a row-stochastic matrix. To prove the claim, it suffices to show that for each $j \in [m]$, the sequence $\{\frac{1}{R^2_j(t)}\}$ is summable. Note that for each $j \in [m]$ and for each t, we have $\bar{R}_j(t+1) = \bar{R}_j(t) + \epsilon_t$ where $\epsilon_t \geq \epsilon > 0$ with probability $p_t \geq p > 0$. Therefore, using the Borel-Cantelli Lemma for adapted processes [19] we have $\{\frac{1}{R^2_j(t)}\}$ is summable which concludes the proof. □

Proof of Lemma 4.4: Fix $i \in [m], j \in [n]$ and $k \in \mathbb{N}$. Let B be the event that at the t_k'th iteration, user reinforces a pair (i, ℓ) for some $\ell \in [n]$. Then, on the complement B^c of B, $P^+_{ij}(\omega) = P_{ij}(\omega)$. Let $B_1 \subseteq B$ be the subset of B such that the pair (i, j) is reinforced and $B_2 = B \setminus B_1$ be the event that some other pair (i, ℓ) is reinforced for $\ell \neq i$.

We note that

$$U^+_{ij} = \frac{S_{ij} + 1}{\sum_{\ell=1}^{n} S_{i\ell} + 1} 1_{B_1} + \frac{S_{ij}}{\sum_{\ell=1}^{n} S_{i\ell} + 1} 1_{B_2} + U_{ij} 1_{B^c}.$$

Therefore, we have

$$\begin{aligned} E(U^+_{ij} \mid \mathcal{F}_{k_t}) &= \pi_i U_{ij} D_{ji} \frac{S_{ij} + 1}{\sum_{\ell=1}^{n} S_{i\ell} + 1} \\ &\quad + \sum_{\ell \neq j} \pi_i U_{i\ell} D_{\ell i} \frac{S_{ij}}{\sum_{\ell'=1}^{n} S_{i\ell'} + 1} + (1-p) U_{ij}, \end{aligned}$$

where $p = U(B \mid \mathcal{F}_{k_t}) = \sum_\ell \pi_i U_{ij} D_{ji}$. Note that $U_{ij} = \frac{S_{ij}}{\sum_{\ell=1}^{n} S_{i\ell}}$ and hence,

$$\begin{aligned} &E(U^+_{ij} \mid \mathcal{F}_t) - U_{ij} = \\ &\qquad \frac{1}{\sum_{\ell'=1}^{n} S_{i\ell'} + 1} \left(\pi_i U_{ij} D_{ji} - \pi_i U_{ij} \sum_{\ell} U_{i\ell} D_{\ell i} \right). \end{aligned}$$

which can be rewritten as in (8). □

Proof of Theorem 4.5: Fix $t = t_k$ for some $k \in \mathbb{N}$. Let $u^+ := u(t+1)$, $u := u(t)$, $u^i := u^i(U(t), D(t))$ and also define $\tilde{S}^i := \sum_{\ell'=1}^{m} S_{i\ell'} + 1$. Then, using the linearity of conditional expectation and Lemma 4.1, we have:

$$\begin{aligned} E(u^+ \mid \mathcal{F}_t) - u &= \sum_{i=1}^{m}\sum_{j=1}^{n} \pi_i D_{ji} \left(E(U^+_{ij} \mid \mathcal{F}_t) - U_{ij} \right) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} \pi_i D_{ji} \frac{\pi_i U_{ij}}{\sum_{\ell'=1}^{m} S_{j\ell'} + 1} \left(D_{ji} - u^i \right) \\ &= \sum_{i=1}^{m} \frac{\pi_i^2}{\tilde{S}^i} \left(\sum_{j=1}^{n} U_{ij} (D_{ji})^2 - (u^i)^2 \right). \end{aligned} \tag{25}$$

Note that U is a row-stochastic matrix and hence, $\sum_{i=1}^{m} U_{ij} = 1$. Therefore, by the Jensen's inequality [19], we have:

$$\sum_{j=1}^{n} U_{ij} (D_{ji})^2 \geq \left(\sum_{j=1}^{n} D_{ji} U_{ij} \right)^2 = (u^i)^2.$$

Replacing this in the right-hand-side of (25), we conclude that $E(u^+ \mid \mathcal{F}_t) - u \geq 0$ and hence, the sequence $\{u(t)\}$ is a submartingale. □

Proof of Corollary 4.6: Note from Theorem 4.3 and 4.5 that the sequence $\{u(t)\}$ satisfies all the conditions of Theorem 4.2. Hence, proven. □

C FURTHER RELATED WORK

Database Interaction Sampling has been used to approximate the results of SQL queries with aggregation functions and achieve the fast response time needed by interactive database interfaces [12, 29]. However, we use sampling techniques to learn the intent behind imprecise point queries and answer them effectively and efficiently.

Reinforcement Learning There is a recent interest in using exploitation-exploration paradigm to improve the understanding of users intents in an interactive document retrieval [24]. The exploitation-exploration trade-off has been also considered in finding keyword queries for data integration [51]. These methods, however, does not consider the impact of user learning throughout the interaction.

Game-theoretic Models in Information Systems Researchers have also leveraged economical models to build query interfaces that return desired results to the users using the fewest possible interactions [53]. In particular, researchers have recently applied game-theoretic approaches to model the actions taken by users and document retrieval systems in a single session [35]. They propose a framework to find out whether the user likes to continue exploring the current topic or move to another topic. We, however, explore the development of common representations of intents between the user and DMBS. We also investigate the interactions that may contain various sessions and topics. Moreover, we focus on structured rather than unstructured data. Avestani et al. have used signaling games to create a shared lexicon between multiple autonomous systems [5]. Our work, however, focuses on modeling users' information needs and development of mutual understanding between users and the DBMS. Moreover, as opposed to the autonomous systems, a DBMS and user may update their information about the interaction in different time scales.

D MORE INFORMATION ABOUT THE SETTING OF EFFICIENCY ANALYSIS

Freebase is built based on the information about entities in the Wikipedia (*wikipedia.org*) articles. Each entity in Freebase database contains the URL of its corresponding article in Wikipedia. For our queries, we have used a sample of Bing (*bing.com*) query log whose relevant answers according to the click-through information, after filtering our noisy clicks, are in the Wikipedia articles [20]. We use two subsets of this sample whose relevant answers are in the *TV-Program* and *Play* databases. The set of queries over *TV-Program* has 621 (459 unique) queries with the average number of 3.65 keywords per query and the one over *Play* has 221 (141 unique) queries with the average number of 3.66 keywords per query. We use the frequencies of queries to calculate the prior probabilities of submission. After submitting each query and getting some results, we simulate user feedback using the relevance information in the Bing query log.

E MITIGATING STARTUP PERIOD

Because our proposed learning algorithm is more exploratory than UCB-1, it may have a longer startup period than UCB-1's. One method is for the DBMS to use a less exploratory learning algorithm, such as UCB-1, at the beginning of the interaction. After a certain number of interactions, the DBMS can switch to our proposed learning algorithm. The DBMS can distinguish the time of switching to our algorithm by observing the amount of positive reinforcement it receives from the user. If the user does not provide any or very small number of positive feedback on the returned results, the DBMS is not yet ready to switch to a relatively more exploratory algorithm. If the DBMS observes a relatively large number of positive feedback on sufficiently many queries, it has already provided a relatively accurate answers to many queries. Finally, one may use a relatively large value of reinforcement in the database learning algorithm at the beginning of the interaction to reduce its degree of exploration. The DBMS may switch to a relatively small value of reinforcement after it observes positive feedback on sufficiently many queries.

We have implemented the latter of these methods by increasing the value of reinforcement by some factor. Figure 3 shows the results of applying this technique in our proposed DBMS learning algorithm over the Yahoo! query workload. The value of reinforcement is initially 3 and 6 times larger than the default value proposed in Section 4 until a threshold satisfaction value is reached, at which point the reinforcement values scales back down to its original rate.

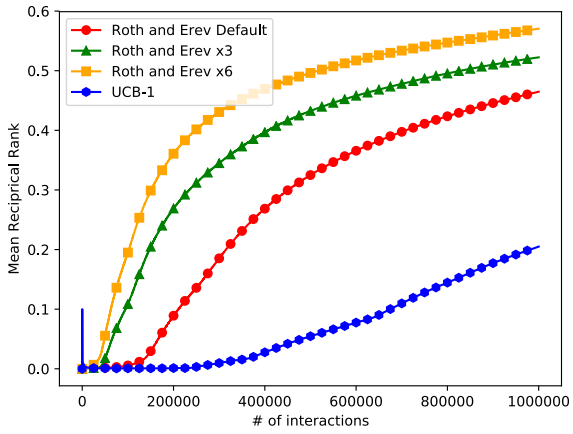

Figure 3: Mean reciprocal rank for 1,000,000 interactions with different degrees of reinforcements

We notice that by increasing the reinforcement value by some factor, the startup period is reduced. However, there are some drawbacks to this method. Although we don't see it here, by increasing the rate of reinforcement in the beginning, some amount of exploration may be sacrificed. Thus more exploitation will occur in the beginning of the series of interactions. This may lead to behavior similar to UCB-1 and perform too much exploitation and not enough exploration. Finding the correct degree of reinforcement is an interesting area for future work.