Example-Driven Query Intent Discovery: Abductive Reasoning using Semantic Similarity

Anna Fariha
College of Information and Computer Sciences
University of Massachusetts Amherst
afariha@cs.umass.edu

Alexandra Meliou
College of Information and Computer Sciences
University of Massachusetts Amherst
ameli@cs.umass.edu

ABSTRACT

Traditional relational data interfaces require precise structured queries over potentially complex schemas. These rigid data retrieval mechanisms pose hurdles for non-expert users, who typically lack language expertise and are unfamiliar with the details of the schema. Query by Example (QBE) methods offer an alternative mechanism: users provide examples of their intended query output and the QBE system needs to infer the intended query. However, these approaches focus on the structural similarity of the examples and ignore the richer context present in the data. As a result, they typically produce queries that are too general, and fail to capture the user’s intent effectively. In this paper, we present SQcID, a system that performs semantic similarity-aware query intent discovery. Our work makes the following contributions: (1) We design an end-to-end system that automatically formulates select-project-join queries in an open-world setting, with optional group-by aggregation and intersection operators; a much larger class than prior QBE techniques. (2) We express the problem of query intent discovery using a probabilistic abduction model, that infers a query as the most likely explanation of the provided examples. (3) We introduce the notion of an abduction-ready database, which precomputes semantic properties and related statistics, allowing SQcID to achieve real-time performance. (4) We present an extensive empirical evaluation on three real-world datasets, including user-intent case studies, demonstrating that SQcID is efficient and effective, and outperforms machine learning methods, as well as the state-of-the-art in the related query reverse engineering problem.

PVLDB Reference Format:

1. INTRODUCTION

Database technology has expanded drastically, and its audience has broadened, bringing on a new set of usability requirements. A significant group of current database users are non-experts, such as data enthusiasts and occasional users. These non-expert users want to explore data, but lack the expertise needed to do so. Traditional database technology was not designed with this group of users in mind, and hence poses hurdles to these non-expert users. Traditional QBE systems for relational databases make a strong and oversimplifying assumption in modeling user intent: they implicitly treat the structural similarity and data content of the example tuples as the only factors specifying query intent. As a result, they consider all queries that contain the provided example tuples in their result set as equally likely to represent the desired intent. This ignores the richer context in the data that can help identify the intended query more effectively.

Example 1.1. In Figure 1, the relations academics and research store information about CS researchers and their research interests. Given the user-provided set of examples \{Dan Suciu, Sam Madden\}, a human can posit that the user is likely looking for data management researchers. However, a QBE system, that looks for queries based only on the structural similarity of the examples, produces \(Q_1\) to capture the query intent, which is too general:

\[Q_1: \text{SELECT name FROM academics WHERE research.aid = academics.id AND research.interest = 'data management'}\]

In fact, the QBE system will generate the same generic query \(Q_1\) for any set of names from the relation academics. Even though the intended semantic context is present in the data (by associating academics with research interest information using the relation research), existing QBE systems fail to capture it. The more specific query that better represents the semantic similarity among the example tuples is \(Q_2\):

\[Q_2: \text{SELECT name FROM academics, research WHERE research.aid = academics.id AND research.interest = 'data management'}\]

Unfortunately, traditional QBE systems [49, 46, 14] for relational databases make a strong and oversimplifying assumption in modeling user intent: they implicitly treat the structural similarity and data content of the example tuples as the only factors specifying query intent. As a result, they consider all queries that contain the provided example tuples in their result set as equally likely to represent the desired intent. This ignores the richer context in the data that can help identify the intended query more effectively.
Example 1.1 shows how reasoning about the semantic similarity of the example tuples can guide the discovery of the correct query structure (join of the academic and research tables), as well as the discovery of the likely intent (research interest in data management).

We can often capture semantic similarity through direct attributes of the example tuples. These are attributes associated with a tuple within the same relation, or through simple key-foreign key joins (such as research interest in Example 1.1). Direct attributes capture intent that is explicit, precisely specified by the particular attribute values. However, sometimes query intent is more vague, and not expressible by explicit semantic similarity alone. In such cases, the semantic similarity of the example tuples is implicit, captured through deeper associations with other entities in the data (e.g., type and quantity of movies an actor appears in).

Example 1.2. The IMDb dataset contains a wealth of information related to the movies and entertainment industry. We query the IMDb dataset (Figure 2) with a QBE system, using two different sets of examples:

ET1={Arnold Schwarzenegger, Sylvester Stallone, Dwayne Johnson} ET2={Eddie Murphy, Jim Carrey, Robin Williams}

ET1 contains the names of three actors from a public list of “physically strong” actors; ET2 contains the names of three actors from a public list of “funny” actors. ET1 and ET2 represent different query intents (strong actors and funny actors, respectively), but a standard QBE system produces the same generic query for both:

Q3: SELECT person.name FROM person

Explicit semantic similarity cannot capture these different intents, as there is no attribute that explicitly labels an actor as “strong” or “funny”. Nevertheless, the database encodes these associations implicitly, in the number and type of movies an actor appears in. Standard QBE systems typically produce queries that are too general, and fail to capture nuanced query intents, such as the ones in Examples 1.1 and 1.2. Some prior approaches attempt to refine the queries based on additional, external information, such as external ontologies [36], provenance information of the example tuples [14], and user feedback on multiple (typically a large number) system-generated examples [11, 35, 16]. Other work relies on a closed-world assumption to produce more expressive queries [35, 54, 62] and thus requires complete examples of input databases and output results. Providing such external information is typically complex and tedious for a non-expert.

In contrast with prior approaches, we propose a method and present an end-to-end system for discovering query intent effectively and efficiently, in an open-world setting, without the need for additional external information, beyond the initial set of example tuples. SQUID, our semantic similarity-aware query intent discovery framework [22], relies on two key insights: (1) It exploits the information and associations already present in the data to derive the explicit and implicit similarities among the provided examples. (2) It identifies the significant semantic similarities among them using abductive reasoning, a logical inference mechanism that aims to derive a query as the simplest and most likely explanation for the observed results (example tuples). We explain how SQUID uses these insights to handle the challenging scenario of Example 1.2.

Example 1.3. We query the IMDb dataset with SQUID, using the example tuples in ET2 (Example 1.2). SQUID discovers the following semantic similarities among the examples: (1) all are Male, (2) all are American, and (3) all appeared in more than 40 Comedy movies. Out of these properties, Male and American are very common in the IMDb database. In contrast, a very small fraction of persons in the dataset are associated with such a high number of Comedy movies; this means that it is unlikely for this similarity to be coincidental, as opposed to the other two. Based on abductive reasoning, SQUID selects the third semantic similarity as the best explanation of the observed example tuples, and produces the query:

Q4: SELECT person.name FROM person, castinfo, movietogenre, genre WHERE person.id = castinfo.person_id AND castinfo.movie_id = movietogenre.movie_id AND movietogenre.genre_id = genre.id AND genre.name = 'Comedy' GROUP BY person.id HAVING count(*) >= 40

In this paper, we make the following contributions:

- We design an end-to-end system, SQOLID, that automatically formulates select-project-join queries with optional group-by-agggregation and intersection operators (SPIoJ) based on few user-provided example tuples. SQUID does not require the users to have any knowledge of the database schema or the query language. In contrast with existing approaches, SQUID does not need any additional user-provided information, and achieves very high precision with very few examples in most cases.
• SQUID infers the semantic similarity of the example tuples, and models query intent using a collection of basic and derived semantic property filters (Section 3). Prior work has explored the use of semantic similarity in knowledge graph retrieval tasks [63, 41, 28]. However, these prior systems do not directly apply to the relational domain, and do not model implicit semantic similarities, derived from aggregating properties of affiliated entities (e.g., number of comedy movies an actor appeared in).

• We express the problem of query intent discovery using a probabilistic abduction model (Section 4). This model allows SQUID to identify the semantic property filters that represent the most probable intent given the examples.

• SQUID achieves real-time performance through an offline strategy that pre-computes semantic properties and related statistics to construct an abduction-ready database (Section 5). During the online phase, SQUID consults the abduction-ready database to derive relevant semantic property filters, based on the provided examples, and applies abduction to select the optimal set of filters towards query intent discovery (Section 6). We prove the correctness of the abduction algorithm in Theorem 1.

• Our empirical evaluation includes three real-world datasets, 41 queries covering a broad range of complex intents and structures, and three case studies (Section 7). We further compare with TALOS [53], a state-of-the-art system that captures very expressive queries, but in a closed-world setting. We show that SQUID is more accurate at capturing intent and produces better queries, often reducing the number of predicates by orders of magnitude. We also empirically show that SQUID outperforms a semi-supervised machine learning system [19], which learns classification models from positive examples and unlabeled data.

2. SQUID OVERVIEW

In this section, we first discuss the challenges in example-driven query intent discovery and highlight the shortcomings of existing approaches. We then formalize the problem of query intent discovery using a probabilistic model and describe how SQUID infers the most likely query intent using abductive reasoning. Finally, we present the system architecture for SQUID, and provide an overview of our approach.

2.1 The Query Intent Discovery Problem

SQUID aims to address three major challenges that hinder existing QBE systems:

Large search space. Identifying the intended query given a set of example tuples can involve a huge search space. Aside from enumerating the candidate queries, validating them is expensive, as it requires executing the queries over potentially very large data. Existing approaches limit their search space in three ways: (1) They often focus on project-join (PJ) queries only. Unfortunately, ignoring selections severely limits the applicability and practical impact of these solutions. (2) They assume that the user provides a large number of examples or interactions, which is often unreasonable in practice. (3) They make a closed-world assumption, thus needing complete sets of input data and output results. In contrast, SQUID focuses on a much larger and more expressive class of queries, select-project-join queries with optional group-by aggregation and intersection operators (SPJQ), and is effective in the open-world setting with very few examples.

Distinguishing candidate queries. In most cases, a set of example tuples does not uniquely identify the target query, i.e., there are multiple valid queries that contain the example tuples in their results. Most existing QBE systems do not distinguish among the valid queries [49] or only rank them according to the degree of input containment, when the example tuples are not fully contained by the query output [46]. In contrast, SQUID exploits the semantic context of the example tuples and ranks the valid queries based on a probabilistic abduction model of query intent.

Large search space.

Complex intent. A user’s information need is often more complex than what is explicitly encoded in the database schema (e.g., Example 1.2). Existing QBE solutions focus on the query structure and are thus ill-equipped to capture nuanced intent. While SQUID still produces a structured query in the end, its objectives focus on capturing the semantic similarity of the examples, both explicit and implicit. SQUID thus draws a contrast between the traditional query-by-example problem, where the query is assumed to be the hidden mechanism behind the provided examples, and the query intent discovery problem that we focus on in this work.

We proceed to formalize the problem of query intent discovery. We use $D$ to denote a database, and $Q(D)$ to denote the set of tuples in the result of query $Q$ operating on $D$.

**Definition 2.1** (Query Intent Discovery). For a database $D$ and a user-provided example tuple set $E$, the query intent discovery problem is to find an SPJQ query $Q$ such that:

- $E \subseteq Q(D)$
- $Q = \text{argmax}_Q \text{Pr}(Q|E)$

More informally, we aim to discover an SPJQ query $Q$ that contains $E$ within its result set and maximizes the query posterior, i.e., the conditional probability $\text{Pr}(Q|E)$.

2.2 Abductive Reasoning

SQUID solves the query intent discovery problem (Definition 2.1) using abduction. Abduction or abductive reasoning [40, 30, 10, 5] refers to the method of inference that finds the best explanation (query intent) of an often incomplete observation (example tuples). Unlike deduction, in abduction, the premises do not guarantee the conclusion. So, a deductive approach would produce all possible queries that contain the example tuples in their results, and it would guarantee that the intended query is one of them. However, the set of valid queries is typically extremely large, growing exponentially with the number of properties and the size of the data domain. In our work, we model query intent discovery as an abduction problem and apply abductive inference to discover the most likely query intent. More formally, given two possible candidate queries, $Q$ and $Q'$, we infer $Q$ as the intended query if $\text{Pr}(Q|E) > \text{Pr}(Q'|E)$.

**Example 2.1.** Consider again the scenario of Example 1.1. SQUID identifies that the two example tuples share the semantic context interest $=$ data management. $Q_1$ and $Q_2$ both contain the example tuples in their result set. However, the probability that two tuples
drawn randomly from the output of Q1 would display the identified semantic context is low \((\frac{2}{9})^2 \approx 0.18\) in the data excerpt. In contrast, the probability that two tuples drawn randomly from the output of Q2 would display the identified semantic context is high \((1.0)\). Assuming that Q1 and Q2 have equal priors \(Pr(Q1) = Pr(Q2)\), then from Bayes’ rule \(Pr(Q2|E) > Pr(Q1|E)\).

### 3. Modeling Query Intent

SQ1ID’s core task is to infer the proper SPJ query on the αDB. We model an SPJ query as a pair of a base query and a set of semantic property filters: \(Q^\varphi = (Q^\varphi, \varphi)\). The base query \(Q^\varphi\) is a project-join query that captures the structural aspect of the example tuples. SQ1ID can handle examples with multiple attributes, but, for ease of exposition, we focus on example tuples that contain a single attribute of a single entity (name of person). In contrast to existing approaches that derive PJ queries from example tuples, the base query in SQ1ID does not need to be minimal with respect to the number of joins: While a base query on a single relation with projection on the appropriate attribute (e.g., Q1 in Example 1.1) would capture the structure of the examples, the semantic context may rely on other relations (e.g., research, as in Q2 of Example 1.1). Thus, SQ1ID considers any number of joins among αDB relations for the base query, but limits these to key-foreign key joins.

We discuss a simple method for deriving the base query in Section 6.2. SQ1ID’s core challenge is to infer \(\varphi\), which denotes a set of semantic property filters that are added as conjunctive selection predicates to \(Q^\varphi\). The base query and semantic property filters for Q2 of Example 1.1 are:

\[
Q^\varphi = \text{SELECT name FROM academics, research WHERE research.aid = academics.id} \\
\varphi = \{\text{research.interested} = \text{‘data management’}\}
\]

### 3.1 Semantic Properties and Filters

Semantic properties encode characteristics of an entity, and are of two types: (1) A basic semantic property is affiliated with an entity directly. In the IMDb schema of Figure 2, gender=Male is a basic semantic property of a person. (2) A derived semantic property of an entity is an aggregate over a basic semantic property of an associated entity. In Example 2.2, the number of movies of a particular genre that a person appeared in is a derived semantic property for person. We represent a semantic property \(p\) of an entity from a relation \(R\) as a triple \(⟨A, V, θ⟩\). In this notation, \(V\) denotes a value\(^3\) or a value range for attribute \(A\) associated with entities in \(R\). The association strength parameter \(θ\) quantifies how strongly an entity is associated with the property. It corresponds to a threshold on derived semantic properties (e.g., the number of comedies an actor appeared in); it is not defined for basic properties (\(θ = \perp\)).

A semantic property filter \(φ_ϕ\) is a structured language representation of the semantic property \(p\). In the data of Figure 6, the filters \(φ_ϕ(\text{gender} = \text{‘Male’}, \text{‘age’} = \{20, 30, 40\})\) and \(φ_ϕ(\text{genre} = \text{‘Comedy’}, \text{‘age’} = \{20, 30, 40\})\) represent two basic semantic properties on gender and age, respectively. Expressed in relational algebra, filters on basic semantic properties map to standard selection predicates, e.g., \(σ_{\text{gender} = \text{‘Male’}(\text{person})\} \text{ and } σ_{\text{age} \leq 30(\text{person})}\). For derived properties, filters specify conditions on the association across different entities. In Example 2.2, for person entities, the filter \(φ_ϕ(\text{genre} = \text{‘Comedy’}, \text{‘count’} \geq 30(\text{person}))\) denotes the property of a person being associated with at least 30 movies with the basic property genre=Comedy. In relational algebra, filters on derived properties map to selection predicates over derived relations in the αDB, e.g., \(σ_{\text{genre} = \text{‘Comedy’}, \text{‘count’} \geq 30(\text{person})}\).

### 3.2 Filters and Example Tuples

To construct \(Q^\varphi\), SQ1ID needs to infer the proper set of semantic property filters given a set of example tuples. Since all example tuples should be in the result of \(Q^\varphi\), \(\varphi\) cannot contain filters that the example tuples do not satisfy. Thus, we only consider valid \(\varphi\) filters that map to selection predicates that all examples satisfy.

---

\(^3\)SQ1ID can support disjunction for categorical attributes (e.g., gender=Male or gender=Female), so \(V\) could be a set of values. However, for ease of exposition we keep our examples limited to properties without disjunction.
Definition 3.1 (Filter validity). Given a database $D$, an example tuple set $E$, and a base query $Q^*$, a filter $\phi$ is valid if and only if $Q^*(\phi) \geq E$, where $Q^*(\phi) = (Q^*, \{\phi\})$.

Figure 6 shows a set of example tuples over the relation person. Given the base query $Q^* = \text{select name from person}$, the filters $\phi_{\text{gender,Male}_1}$ and $\phi_{\text{age,}[50,90],1}$ on relation person are valid, because all of the example entities of Figure 6 are Male and fall in the age range [50, 90].

Lemma 3.1. (Validity of conjunctive filters). The conjunction $(\phi_1 \land \phi_2 \land \ldots)$ of a set of filters $\Phi = \{\phi_1, \phi_2, \ldots\}$ is valid, i.e., $Q^*(\Phi) \geq E$, if and only if $\forall \phi_i \in \Phi: \phi_i$ is valid.

Relaxing a filter (loosening its conditions) preserves validity. For example, if $\phi_{\text{age,}[50,90],1}$ is valid, then $\phi_{\text{age,}[40,120],1}$ is also valid. Out of all valid filters, SQ1ID focuses on minimal valid filters, which have the tightest bounds.

Definition 3.2 (Filter minimality). A basic semantic property filter $\phi_{(A,V,\theta)}$ is minimal if it is valid, and $\forall V' \subseteq V, \phi_{(A,V',\theta)}$ is not valid. A derived semantic property filter $\phi_{(A,V,\theta)}$ is minimal if it is valid, and $\forall V > 0, \phi_{(A,V,\theta)}$ is not valid.

In the example of Figure 6, $\phi_{\text{age,}[50,90],1}$ is a minimal filter and $\phi_{\text{age,}[40,90],1}$ is not.

4. PROBABILISTIC ABDUCTION MODEL

We now revisit the problem of Query Intent Discovery (Definition 2.1), and recast it based on our model of query intent (Section 3). Specifically, Definition 2.1 aims to discover an $\text{SPJ}_A$ query $Q_2$; this is reduced to an equivalent SPJ query $Q^*$ on the oDB (as in Example 2.2). SQ1ID’s task is to find the query $Q^*$ that maximizes the posterior probability $Pr(Q^*/E)$, for a given set $E$ of example tuples. In this section, we analyze the probabilistic model to compute this posterior, and break it down to three components.

4.1 Notations and Preliminaries

Semantic context $\psi$. Observing a semantic property in a set of 10 examples is more significant than observing the same property in a set of 2 examples. We denote this distinction with the semantic context $\psi = (A,V,\theta)$, which encodes the size of the set of example tuples $E$.

Candidate SPJ query $Q^*$. Let $\Phi = \{\phi_1, \phi_2, \ldots\}$ be the set of minimal valid filters, where $\phi_i$ encodes the semantic context $\psi_i$. Our goal is to identify the subset of filters in $\Phi$ that best captures the query intent. A set of filters $\varphi \subseteq \Phi$ defines a candidate query $Q^* = (Q^*, \varphi)$, and $Q^*(\varphi) \geq E$ (from Lemma 3.1).

Filter event $\phi$. A filter $\phi \in \Phi$ may or may not appear in a candidate query $Q^*$. With slight abuse of notation, we denote the filter's presence ($\phi \in \varphi$) with $\phi$ and its absence ($\phi \notin \varphi$) with $\bar{\phi}$. We use $\tilde{\phi}$ to represent the occurrence event of $\phi$ in $Q^*$.

Thus: $\tilde{\phi} = \begin{cases} \phi & \text{if } \phi \in \varphi \\ \bar{\phi} & \text{if } \phi \notin \varphi \end{cases}$.

4.2 Modeling Query Posterior

We first analyze the probabilistic model for a fixed base query $Q^*$ and then generalize the model in Section 4.3. We use $Pr_*(a)$ as a shorthand for $Pr(a|Q^*)$. We model the query posterior $Pr_*(Q^*/E)$, using Bayes’ rule:

$$Pr_*(Q^*/E) = \frac{Pr_*(E|Q^*)Pr_*(Q^*)}{Pr_*(E)}$$

By definition, $Pr_*(X|E) = 1$; therefore:

$$Pr_*(Q^*/E) = \frac{Pr_*(E|X,Q^*)Pr_*(Q^*)}{Pr_*(E)} = \frac{Pr_*(E|X,Q^*)Pr_*(X|Q^*)Pr_*(Q^*)}{Pr_*(E)}$$

Using the fact that $Pr_*(X|E) = 1$ and applying Bayes’ rule on the prior $Pr_*(E)$, we get:

$$Pr_*(Q^*/E) = \frac{Pr_*(E|X,Q^*)Pr_*(X|Q^*)Pr_*(Q^*)}{Pr_*(E)}$$

Finally, $E$ is conditionally independent of $Q^*$ given the semantic context $X$; i.e., $Pr_*(E|X,Q^*) = Pr_*(E|X)$. Thus:

$$Pr_*(Q^*/E) = \frac{Pr_*(E|X,Q^*)Pr_*(Q^*)}{Pr_*(E)}$$

In Equation 1, we have modeled the query posterior in terms of three components: (1) the semantic context prior $Pr_*(X)$, (2) the query prior $Pr_*(Q^*)$, and (3) the semantic context posterior, $Pr_*(X|Q^*)$. We proceed to analyze each of these components.

4.2.1 Semantic Context Prior

The semantic context prior $Pr_*(X)$ denotes the probability that any set of example tuples of size $|E|$ exhibits the semantic context $X$. This probability is not easy to compute analytically, as it involves computing a marginal over a potentially infinite set of candidate queries. In this work, we model the semantic context prior as proportional to the selectivity $\psi(\Phi)$ of $\Phi = \{\phi_1, \phi_2, \ldots\}$, where $\phi_i \in \Phi$ is a filter that encodes context $x_i \in X$:

$$Pr_*(X) \propto \psi(\Phi)$$

Selectivity $\psi(\Phi)$. The selectivity of a filter $\phi$ denotes the portion of tuples from the result of the base query $Q^*$ that satisfy $\phi$:

$$\psi(\phi) = \frac{|Q^*(\phi)|}{|Q^*(\Phi)|}$$

Similarly, for a set of filters $\Phi$, $\psi(\Phi) = \frac{|Q^*(\Phi)|}{|Q^*(\Phi)|}$. Intuitively, a selectivity value close to 1 means that the filter is not very selective and most tuples satisfy the filter; selectivity value close to 0 means that the filter is very selective and few tuples satisfy the filter.
0 denotes that the filter is highly selective and rejects most of the tuples. For example, in Figure 6, denotes more selective than . With selectivities and , respectively.

Selectivity captures the rarity of a semantic context: uncommon contexts are present in fewer tuples and thus appear in the output of fewer queries. Intuitively, a rare context has lower prior probability that the filter is highly selective and rejects most of the tuples. For example, if a user is interested in actors and , then this family of filters is less likely than .

\[ \Pr(\Phi) \text{ is included in (if } \phi \notin \Phi \text{)} \]

We describe these parameters next.

**Domain selectivity impact \( \delta(\phi) \)**. Intuitively, a filter that covers a large range of values in an attribute’s domain is unlikely to be part of the intended query. For example, if a user is interested in actors of a certain age group, then the group is more likely to be narrow (as for ) than broad (as for ). We penalize broad filters with the parameter \( \delta \in [0, 1] \); \( \delta(\phi) = 1 \) for filters that do not exceed a predefined ratio in the coverage of their domain, and decreases for filters that exceed this threshold.

**Association strength impact \( \alpha(\phi) \)**. Intuitively, a derived filter with low association strength is unlikely to appear in the intended query, as the filter denotes a weak association with the relevant entities. For example, is less likely than to represent a query intent. We label filters with \( \theta \) lower than a threshold \( \tau_a \) as insignificant, and set \( \alpha(\phi) = 0 \). All other filters, including basic filters, have \( \alpha(\phi) = 1 \).

**Outlier impact \( \lambda(\phi) \)**. While \( \alpha(\phi) \) characterizes the impact of association strength on a filter individually, \( \lambda(\phi) \) characterizes its impact in consideration with other derived filters over the same attribute. Figure 8 demonstrates two cases of derived filters on the same attribute (genre), corresponding to two different sets of example tuples. In Case A, \( \phi_1 \) and \( \phi_2 \) are more significant than the other filters of the same family (higher association strength). Intuitively, this corresponds to the intent to retrieve actors who appeared in mostly comedy and science fiction movies. In contrast, Case B does not have filters that stand out, as all have similar association strengths: The actors in this example set are not strongly associated with particular genres, and thus, intuitively, this family of filters is not relevant to the query intent.

We model the outlier impact \( \lambda(\phi) \) of a filter using the skewness of the association strength distribution within the family of derived filters sharing the same attribute. Our assumption is that highly-skewed, heavy-tailed distributions (Case A) are likely to contain the significant (intended) filters as outliers. We set \( \lambda(\phi) = 1 \) for a derived filter whose association strength is an outlier in the association strength distribution of filters of the same family. We also set \( \lambda(\phi) = 1 \) for basic filters. All other filters get \( \lambda(\phi) = 0 \).

\[ \Pr(\phi) = \Pr(\phi, \text{Comedy}) \]

\[ \Pr(\phi) = \Pr(\phi, \text{SciFi}) \]

\[ \Pr(\phi) = \Pr(\phi, \text{Drama}) \]

\[ \Pr(\phi) = \Pr(\phi, \text{Action}) \]

\[ \Pr(\phi) = \Pr(\phi, \text{Thriller}) \]

**Semantic Context Posterior**

The semantic context posterior \( \Pr(\phi|Q) \) is the probability that a set of example tuples of size \( |E| \), sampled from the output of a particular query \( Q \), exhibits the set of semantic contexts \( \phi \):

\[ \Pr(\phi|Q) = \Pr(\phi_1, \phi_2, \ldots ) \]

Recall that \( \phi \) encodes the semantic context \( x_i \) (Section 4.1). We assume that \( x_i \) is conditionally independent of any \( \phi_i, i \neq j \), given \( \phi_i \) (this always holds for \( \phi_i = \phi_1 \)):

\[ \Pr(\phi_i|Q) = \prod_{i=1}^n \Pr(\phi_i) \]

For each \( x_i \), we compute \( \Pr(\phi_i, \phi_i|Q) \) based on the state of the filter event \( \phi_i = \phi_1 \) or \( \phi_i = \phi_2 \): \( \Pr(\phi_i, \phi_i|Q) \). By definition, all tuples in \( Q \) exhibit the property of \( x_i \). Hence, \( \Pr(\phi_i, \phi_i|Q) = 1 \).

\[ \Pr(\phi|x_i) = \frac{\prod_{i=1}^n \Pr(\phi_i) \Pr(\phi_i|x_i)}{\prod_{i=1}^n \Pr(\phi_i) \Pr(\phi_i|x_i)} \]

Using Equations (1)–(4), we derive the final form of the query posterior (where \( K \) is a normalization constant):

\[ \Pr(Q|\phi) = \frac{K}{\psi(\phi)} \prod_{\phi_i \in \phi} \left( \Pr(\phi_i) \Pr(\phi_i|x_i) \right) \]

\[ \Pr(Q|\phi) = \frac{K}{\psi(\phi)} \prod_{\phi_i \in \phi} \left( \Pr(\phi_i) \Pr(\phi_i|x_i) \right) \]

\[ \Pr(Q|\phi) = \frac{K}{\psi(\phi)} \prod_{\phi_i \in \phi} \left( \Pr(\phi_i) \Pr(\phi_i|x_i) \right) \]

**Generalization**

So far, our analysis focused on a fixed base query. Given an SPJ query \( Q \), the underlying base query \( Q^* \) is deterministic, i.e., \( \Pr(Q^*|Q) = 1 \). Hence:

\[ \Pr(Q^*|E) = \Pr(Q^*, Q^*|E) = \Pr(Q^*|Q) \]

\[ \Pr(Q^*|E) = \Pr(Q^*|Q) \]

We assume \( \Pr(Q^*|E) \) to be equal for all valid base queries, where \( Q^*(E) \). Then we use \( \Pr(Q^*|E) \) to find the query \( Q \) that maximizes the query posterior \( \Pr(Q|E) \).

**5. Offline Abduction Preparation**

In this section, we discuss system considerations to perform query intent discovery efficiently. SQV/ID employs an offline module that performs several pre-computation steps to make the database abduction-ready. The abduction-ready database (oDB) augments
the original database with derived relations that store associations across entities and precomputes semantic property statistics. Deriving this information is relatively straightforward; the contributions of this section lie in the design of the DB, the information it maintains, and its role in supporting efficient query intent discovery. We describe the three major functions of the DB.

**Entity lookup.** SQUID’s goal is to discover the most likely query, based on the user-provided examples. To do that, it first needs to determine which entities in the database correspond to the examples. SQUID uses a global inverted column index [49], built over all text attributes and stored in the DB, to perform fast lookups, matching the provided example data to entities in the database.

**Semantic property discovery.** To reason about intent, SQUID first needs to determine what makes the examples similar. It looks for semantic properties within entity relations (e.g., gender appears in table person), other relations (e.g., genre appears in a separate table joining with movie through a FK-FK constraint), and other entities (e.g., the number of movies of a particular genre that a person has appeared in). The DB precomputes and stores such derived relations (e.g., persontogene), as these may involve several joins and aggregations and performing them at runtime would be prohibitive. For example, SQUID computes the persontogene relation (Figure 5) and stores it in the DB with the SQL query below:

```sql
Q6: CREATE TABLE persontogene as
(SELECT person_id, genre_id, count(*) as count
FROM castinfo, movietogene
WHERE castinfo.movie_id = movietogene.movie_id
GROUP BY person_id, genre_id)
```

For the DB construction, SQUID only relies on very basic information to understand the data organization. It uses (1) the database schema, including the specification of primary and foreign key constraints, and (2) additional meta-data, which can be provided once by a database administrator, that specify which tables describe entities (e.g., person, movie), and which tables and attributes describe direct properties of entities (e.g., genre, age). SQUID then automatically discovers fact tables, which associate entities and properties, by exploiting the key-foreign key relationships. SQUID also automatically discovers derived properties up to a certain pre-defined depth, using paths in the schema graph, that connect entities to properties. Since the number of possible values for semantic properties is typically very small and remains constant as entities grow, the DB grows linearly with the data size. In our implementation, we restrict the derived property discovery to the depth of two fact-tables (e.g., SQUID derives persontogene through castinfo and movietogene). SQUID can support deeper associations, but we found these are not common in practice. SQUID generally assumes that different entity types appear in different relations, which is the case in many commonly-used schema types, such as star, galaxy, and fact-constellation schemas. SQUID can perform inference in a denormalized setting, but would not be able to produce and reason about derived properties in those cases.

**Smart selectivity computation.** For basic filters over categorical values, SQUID stores the selectivity for each value. For numeric ranges, SQUID precomputes selectivities $\psi(\phi_{A,\{l,v\}})$ for all $v \in V_A$, where $V_A$ is the set of values of attribute $A$ in the corresponding relation, and $\min_{V_A}$ is the minimum value in $V_A$. The DB can then derive the selectivity of a filter with any value range as:

$$\psi(\phi_{A,\{l,v\}}) = \psi(\phi_{A,\{\min_{V_A}, v\}}) - \psi(\phi_{A,\{\min_{V_A}, l\}})$$

In case of derived semantic properties, SQUID precomputes selectivities $\psi(\phi_{A,v})$ for all $v \in V_A$, where $\Theta_{A,v}$ is the set of values of association strength for the property “$A = v$”.

### 6. QUERY INTENT DISCOVERY

During normal operation, SQUID receives example tuples from a user, consults the DB, and infers the most likely query intent (Definition 2.1). In this section, we describe how SQUID resolves ambiguity in the provided examples, how it derives their semantic context, and how it finally abduces the intended query.

#### 6.1 Entity and Context Discovery

SQUID’s probabilistic abduction model (Section 4) relies on the set of semantic contexts $\mathcal{X}$ and determines which of these contexts are intended vs coincidental, by the inclusion or exclusion of the corresponding filters in the inferred query. To derive the set of semantic contexts from the examples, SQUID first needs to identify the entities in the DB that correspond to the provided examples.

#### 6.1.1 Entity disambiguation

User-provided examples are not complete tuples, but often single-column values that correspond to an entity. As a result, there may be ambiguity that SQUID needs to resolve. For example, suppose the user provides the examples: [Titanic, Pulp Fiction, The Matrix]. SQUID consults the precomputed inverted column index to identify the attributes (movie.title) that contain all the example values, and classifies the corresponding entity (movie) as a potential match. However, while the dataset contains unique entries for Pulp Fiction (1994) and The Matrix (1999), there are 4 possible mappings for Titanic: (1) a 1915 Italian film, (2) a 1943 German film, (3) a 1953 film by Jean Negulesco, and (4) the 1997 blockbuster film by James Cameron.

The key insight for resolving such ambiguities is that the provided examples are more likely to be alike. SQUID selects the entity mappings that maximize the semantic similarities across the examples. Therefore, based on the year and country information, it determines that Titanic corresponds to the 1997 film, as it is most similar to the other two (unambiguous) entities. In case of derived properties, e.g., nationality of actors appearing in a film, SQUID aims to increase the association strength (e.g., the number of such actors). Since the examples are typically few, SQUID can determine the right mappings by considering all combinations.

#### 6.1.2 Semantic context discovery

Once SQUID identifies the right entities, it then explores all the semantic properties stored in the DB that match these entities (e.g., year, genre, etc.). Since the DB precomputes and stores the derived properties, SQUID can produce all the relevant properties using queries with at most one join. For each property, SQUID produces semantic contexts as follows:

**Basic property on categorical attribute.** If all examples in $E$ contain value $v$ for the property of attribute $A$, SQUID produces the semantic context $\langle \{A,v\}, \emptyset, |E| \rangle$. For example, a user provides three movies: Dunkirk, Logan, and Taken. The attribute genre corresponds to a basic property for movies, and all these movies share the values, Action and Thriller, for this property. SQUID generates two semantic contexts: $\langle \{\text{genre,Action}, \perp\}, 3 \rangle$ and $\langle \{\text{genre, Thriller}, \perp\}, 3 \rangle$

**Basic property on numerical attribute.** If $v_{\text{min}}$ and $v_{\text{max}}$ are the minimum and maximum values, respectively, that the examples in $E$ demonstrate for the property of attribute $A$, SQUID creates a semantic context on the range $\langle [v_{\text{min}}, v_{\text{max}}], \perp, |E| \rangle$. For example, if $E$ contains three
Algorithm 1: Query Abduction ($E, Q^*, \Phi$)

Input: set of entities $E$, base query $Q^*$, set of minimal valid filters $\Phi$
Output: $Q^\prime$ such that $Pr_{\phi}(Q^\prime | E)$ is maximized

1. $X = \{x_1, x_2, \ldots\}$ // semantic contexts in $E$
2. $\phi = \emptyset$
3. foreach $\phi_i \in \Phi$ do
4.    $include_{\phi_i} = Pr_{\phi_i}(\phi_i) Pr_{\phi_i}(x_i | \phi_i)$ // from Equation (5)
5.    $exclude_{\phi_i} = Pr_{\phi_i}(\phi_i) Pr_{\phi_i}(x_i | \phi_i)$ // from Equation (5)
6.    If $include_{\phi_i} > exclude_{\phi_i}$ then
7.       $\phi = \phi \cup \{\phi_i\}$
8. return $Q^\prime$

Persons with ages 45, 50, and 52, SQUID will produce the context $\langle\text{(age}, [45, 52], \bot), 3\rangle$.

Derived property. If all examples in $E$ contain value $v$ for the derived property of attribute $A$, SQUID produces the semantic context $\langle(A, v, \theta_{\text{min}}), |E|\rangle$, where $\theta_{\text{min}}$ is the minimum association strength for the value $v$ among all examples. For example, if $E$ contains two persons who have appeared in 3 and 5 Comedy movies, SQUID will produce the context $\langle(\text{genre}, \text{Comedy}, 3), 2\rangle$.

6.2 Query Abduction

SQUID starts abduction by constructing a base query that captures the structure of the example tuples. Once it identifies the entity and attribute that matches the examples (e.g., person.name), it forms the minimal PJ query (e.g., SELECT name FROM person). It then iterates through the discovered semantic contexts and appends the corresponding relations to the FROM clause and the appropriate key-foreign key join conditions in the WHERE clause. Since the $\sigma$DB precomputes and stores the derived relations, each semantic context will add at most one relation to the query.

The number of candidate base queries is typically very small. For each base query $Q^*$, SQUID abduces the best set of filters $\phi \subseteq \Phi$ to construct SPJ query $Q^\prime$, by augmenting the WHERE clause of $Q^*$ with the corresponding selection predicates. (SQUID also removes from $Q^\prime$ any joins that are not relevant to the selected filters $\phi$.)

While the number of candidate SPJ queries grows exponentially in the number of minimum valid filters $\Phi$, Algorithm 1 returns the query $Q^\prime$ that maximizes the query posterior probability (lines 6-7). Our query abduction algorithm has $O(|\Phi|)$ time complexity and is guaranteed to find the query $Q^\prime$ that maximizes the query posterior.

Theorem 1. Given a base query $Q^*$, a set of examples $E$, and a set of minimal valid filters $\Phi$, Algorithm 1 returns the query $Q^\prime$, where $\phi \subseteq \Phi$, such that $Pr_{\phi}(Q^\prime | E)$ is maximized.\footnote{Proof is provided in our technical report [21].}

7. EXPERIMENTS

In this section, we present an extensive experimental evaluation of SQUID over three real-world datasets, with a total of 41 benchmark queries of varying complexities. Our results show that SQUID is scalable and effective, even with a small number of example tuples. Our evaluation includes qualitative case studies over real-world user-generated examples, which demonstrate that SQUID excels in inferring the query intent of real-world users. We further demonstrate that when used as a query-reverse-engineering system in a closed-world setting SQUID outperforms the state-of-the-art. Finally, we show that SQUID is superior to semi-supervised PU-learning in terms of both efficiency and effectiveness.
7.2 Scalability

In our first set of experiments, we examine the scalability of SQuID against increasing number of examples and varied dataset sizes. Figure 9(a) displays the abduction time for the IMDb and DBLP datasets as the number of provided examples increases, averaged over all benchmark queries in each dataset. Since SQuID retrieves semantic properties and computes context for each example, the runtime increases linearly with the number of examples, which is what we observe in practice.

Figure 9(b) extends this experiment to datasets of varied sizes. We generate three alternative versions of the IMDb dataset: (1) sm-IMDb (75 MB), a downsized version that keeps 10% of the original data; (2) bs-IMDb (1330 MB), doubles the entities of the original dataset and creates associations among the duplicate entities (person and movie) by replicating their original associations; (3) bd-IMDb (1926 MB), is the same as bs-IMDb but also introduces associations between the original entities and the duplicates, creating denser connections. SQuID’s runtime increases for all datasets with the number of examples, and, predictably, larger datasets face longer abduction times. Query abduction involves point queries to retrieve semantic properties of the entities, using B-tree indexes. As the data size increases, the runtime of these queries grows logarithmically. SQuID is slower on bd-IMDb than on bs-IMDb: both datasets include the same entities, but bd-IMDb has denser associations, which results in additional derived semantic properties.

7.3 Abduction Accuracy

Intuitively, with a larger number of examples, abduction accuracy should increase: SQuID has access to more samples of the query output, and can more easily distinguish coincidental from intended similarities. Figure 10 confirms this intuition, and precision, recall, and f-score increase, often very quickly, with the number of examples for most of our benchmark queries. We discuss here a few particular queries.

Details of the data generation process are in our technical report [21].
We created a list of names of 211 "funny actors", collected from human-created public lists and Google Knowledge Graph (sources are in our technical report [21]), and used these names as examples of the query intent "funny actors." Figure 13(a) demonstrates the accuracy of the abduced query over a varying number of examples. Each data point is an average across 10 different random samples of example sets of the corresponding size. For this experiment, we tuned SQuID to normalize the association strength, which means that the relevant predicate would consider the fraction of movies in an actor's portfolio classified as comedies, rather than the absolute number.

2000s Sci-Fi movies (IMDb). We used a user-created list of 165 Sci-Fi movies released in 2000s as examples of the query intent "2000s Sci-Fi movies". Figure 13(b) displays the accuracy of the abduced query, averaged across 10 runs for each example set size.

Prolific database researchers (DBLP). We collected a list of database researchers who served as chairs, group leaders, or program committee members in SIGMOD 2011–2015 and selected the top 30 most prolific. Figure 13(c) displays the accuracy of the abduced query averaged, across 10 runs for each example set size.

Analysis. In our case studies there is no (reasonable) SQL query that models the intent well and produces an output that exactly matches our lists. Public lists have biases, such as not including less well-known entities even if these match the intent. In our prolific researchers use case, some well-known and prolific researchers may happen to not serve in service roles frequently, or their commitments may be in venues we did not sample. Therefore, it is not possible to achieve high precision, as the data is bound to contain and retrieve entities that don’t appear on the lists, even if the query is a good match for the intent. For this reason, our precision numbers in the case studies are low. However our recall rises quickly with enough examples, which indicates that the abduced queries converge to the correct intent.

7.5 Query Reverse Engineering

We present an experimental comparison of SQuID with TALOS [53], a state-of-the-art Query Reverse Engineering (QRE) system. QRE systems operate in a closed-world setting, assuming that the provided examples comprise the entire query output. In contrast, SQuID assumes an open-world setting, and only needs a few examples. In the closed-world setting, SQuID is handicapped against a dedicated QRE system, as it does not take advantage of the closed-world constraint in its inference.

For this evaluation under the QRE setting, we use the IMDb and DBLP datasets, as well as the Adult dataset, on which TALOS was shown to perform well [53]. For each dataset, we provided the entire output of the benchmark queries as input to SQuID and TALOS. Since there is no need to drop coincidental filters for query reverse engineering, we set the parameters so that SQuID behaves optimistically (e.g., high filter prior, low association strength threshold, etc.). We adopt the notion of instance equivalent query (IEQ) from the QRE literature [53] to express that two queries produce the same set of results on a particular database instance. A QRE task is successful if the system discovers an IEQ of the original query (f-score=1). For the IMDb dataset, SQuID was able to successfully reverse engineer 11 out of 16 benchmark queries. Additionally, in 4 cases where exact IEQs were not abduced, SQuID queries generated output with ≥ 0.98 f-score. SQuID failed only for IQ10, which is a query that falls outside the supported query family, as discussed in Section 7.3. For the DBLP and Adult datasets, SQuID successfully reverse-engineered all benchmark queries.

Comparison with TALOS. We compare SQuID to TALOS on three metrics: number of predicates (including join and selection predicates), query discovery time, and F-score.

Adult. Both SQuID and TALOS achieved perfect f-score on the 20 benchmark queries. Figure 14 compares the systems in terms of the
number of predicates in the queries they produce (top) and query discovery time (bottom). SQuID almost always produces simpler queries, close in the number of predicates to the original query, while TALOS queries contain more than 100 predicates in 20% of the cases. SQuID is faster than TALOS when the input cardinality is low (~100 tuples), and becomes slower for the largest input sizes (> 700 tuples). SQuID was not designed as a QRE system, and in practice, users rarely provide large example sets. SQuID’s focus is on inferring simple queries that model the intent, rather than cover all examples with potentially complex and lengthy queries.

**IMDb.** Figure 15(a) compares the two systems on the 16 benchmark queries of the IMDb dataset. SQuID produces better queries in almost all cases: in all cases, our abduced queries where significantly smaller, and our f-score is higher for most queries. SQuID was also faster than TALOS for most of the benchmark queries. We now delve deeper into some particular cases.

For IQ1 (cast of Pulp Fiction), TALOS produces a query with f-score = 0.7. We attempted to provide guidance to TALOS through a system parameter that specifies which attributes to include in the selection predicates (which would give it an unfair advantage). TALOS first performs a full join among the participating relations (person and castinfo) and then performs classification on the de-normalized table (with attributes person, movie, role). TALOS gives all rows referring to a cast member of Pulp Fiction a positive label (based on the examples), regardless of the movie that row refers to, and then builds a decision tree based on these incorrect labels. This is a limitation of TALOS, which SQuID overcomes by looking at the semantic similarities of the examples, rather than treating them simply as labels.

SQuID took more time than TALOS in IQ4, IQ7, and IQ15. The result sets of IQ4 and IQ15 are large (> 1000), so this is expected. IQ7 retrieves all movie genres without a selection predicate. As a decision tree approach, TALOS has the advantage here, as it stops at the root and does not need to traverse the tree. In contrast, SQuID retrieves all semantic properties of the example tuples only to discover that either there is nothing common among them, or the property is not significant. While SQuID takes longer, it still abduces the correct query. These cases are not representative of QBE scenarios, as users are unlikely to provide large number of example tuples or have very general intents (PJ queries without selection).

**DBLP.** Figure 15(b) compares the two systems on the DBLP dataset. Here, SQuID successfully reverse engineered all five benchmark queries, but TALOS failed to reverse engineer two of them. TALOS also produced very complex queries, with 100 or more predicates for four of the cases. In contrast, SQuID’s abductions were orders of magnitude smaller, on par with the original query. On this dataset, SQuID was slower than TALOS, but not by a lot.

### 7.6 Comparison with learning methods

Query intent discovery can be viewed as a one-class classification problem, where the task is to identify the tuples that satisfy the desired intent. Positive and Unlabeled (PU) learning addresses this problem setting by learning a classifier from positive examples and unlabeled data in a semi-supervised setting. We compare SQuID against an established PU-learning method [19] on 20 benchmark queries of the Adult dataset. The setting of this experiment conforms with the technique’s requirements [19]: the dataset comprises of a single relation and the examples are chosen uniformly at random from the positive data.

Figure 16(a) compares the accuracy of SQuID and PU-learning using two different estimators, decision tree (DT) and random forest (RF). We observe that PU-learning needs a large fraction (> 70%) of the query result to achieve f-score comparable to SQuID. PU-learning favors precision over recall, and the latter drops significantly when the number of examples is low. In contrast, SQuID achieves robust performance, even with few examples, because it can encode problem-specific assumptions (e.g., that there exists an underlying SQL query that models the intent, that some filters are more likely than other filters, etc.); this cannot be done in straightforward ways for machine learning methods.
To evaluate scalability, we replicated the Adult dataset, with a scale factor up to 10x. Figure 16(b) shows that PU-learning becomes significantly slower than SQU\textsuperscript{ID} as the data size increases, whereas SQU\textsuperscript{ID}'s runtime performance remains largely unchanged. This is because SQU\textsuperscript{ID} does not directly operate on the data outside of the examples (unlabeled data); rather, it relies on the αDB, which contains a highly compressed summary of the semantic property statistics (e.g., filter selectivities) of the data. In contrast, PU-learning builds a new classifier over all of the data for each query intent discovery task. Section 8 provides further discussion on the connections between SQU\textsuperscript{ID} and machine learning approaches.

8. RELATED WORK

Query-by-Example (QBE) was an early effort to assist users without SQL expertise in formulating SQL queries [64]. Existing QBE systems [49, 46] identify relevant relations and joins in situations where the user lacks schema understanding, but are limited to project-join queries. These systems focus on the common structure of the example tuples, and do not try to learn the common semantics as SQU\textsuperscript{ID} does. QPlain [14] uses user-provided provenance of the example tuples to learn the join paths and improve intent inference. However, this assumes that the user understands the schema, content, and domain to provide these provenance explanations, which is often unrealistic for non-experts.

Set expansion is a problem corresponding to QBE in Knowledge Graphs [63, 55, 57]. SPARQLByE [15], built on top of a SPARQL QRE system [4], allows querying RDF datasets by annotated (positive/negative) example tuples. In semantic knowledge graphs, systems address the entity set expansion problem using maximal-aspect-based entity model, semantic-feature-based graph query, and entity co-occurrence information [36, 28, 26, 41]. These approaches exploit the semantic context of the example tuples, but they cannot learn new semantic properties, such as aggregates involving numeric values, that are not explicitly stored in the knowledge graph, and they cannot express derived semantic properties without exploding the graph size. For example, to represent “appearing in more than K comedies”, the knowledge graph would require one property for each possible value of K.

Interactive approaches rely on relevance feedback on system-generated tuples to improve query inference and result delivery [1, 11, 16, 23, 35]. Such systems typically expect a large number of interactions, and are often not suitable for non-experts who may not be sufficiently familiar with the data to provide effective feedback.

Query Reverse Engineering (QRE) [56, 6] is a special case of QBE that assumes that the provided examples comprise the complete output of the intended query. Because of this closed-world assumption, QRE systems can build data classification models on denormalized tables [53], labeling the provided tuples as positive examples and the rest as negative. Such methods are not suitable for our setting, because we operate with few examples, under an open-world assumption. While few QRE approaches [31] relax the closed world assumption (known as the super-set QRE problem) they are also limited to PJ queries similar to the existing QBE approaches. Most QRE methods are limited to narrow classes of queries, such as PJ [61, 31], aggregation without joins [51], or top-k queries [45]. REGAL+[52] handles SPJA queries but only considers the schema of the example tuples to derive the joins and ignores other semantics. In contrast, SQU\textsuperscript{ID} considers joining relations without attributes in the example schema (Example 1.1).

A few QRE methods target expressive SPI queries [62, 54], but they only work for very small databases (< 100 cells), and do not scale to the datasets used in our evaluation. Moreover, the user needs to specify the data in their entirety, thus expecting complete schema knowledge, while SCYTHE [54] also expects user hints towards precise discovery of the constants of the query predicates.

Machine learning methods can model QBE settings as classification problems, and relational machine learning targets relational settings in particular [24]. However, while the examples serve as positive labels, QBE settings do not provide explicit negative examples. Semi-supervised statistical relational learning techniques [58] can learn from unlabeled and labeled data, but require unbiased sample of negative examples. There is no obvious way to obtain such a sample in our problem setting without significant user effort.

Our problem setting is better handled by one-class classification [38, 32], more specifically, Positive and Unlabeled (PU) learning [59, 37, 9, 19, 8, 42], which learns from positive examples and unlabeled data in a semi-supervised setting [13]. Most PU-learning methods assume denormalized data, but relational PU-learning methods do exist. However, all PU-learning methods rely on one or more strong assumptions [9] (e.g., all unlabeled entities are negative [44], examples are selected completely at random [19, 7], positive and negative entities are naturally separable [59, 37, 50], similar entities are likely from the same class [31]). These assumptions create a poor fit for our problem setting where the example set is very small, it may exhibit user biases, response should be real-time, and intents may involve deep semantic similarity.

Other approaches that assist users in query formulation involve query recommendation based on collaborative filtering [18], query autocompletion [34], and query suggestion [20, 17, 29]. Another approach to facilitating data exploration is keyword-based search [3, 27, 60]. User-provided examples and interactions appear in other problem settings, such as learning schema mappings [48, 47, 12]. The query likelihood model in IR [39] resembles our technique, but does not exploit the similarity of the input entities.

9. SUMMARY AND FUTURE DIRECTIONS

In this paper, we focused on the problem of query intent discovery from a set of example tuples. We presented SQU\textsuperscript{ID}, a system that performs query intent discovery effectively and efficiently, even with few examples in most cases. The insights of our work rely on exploiting the rich information present in the data to discover similarities among the provided examples, and distinguish between those that are coincidental and those that are intended. Our contributions include a probabilistic abduction model and the design of an abduction-ready database, which allow SQU\textsuperscript{ID} to capture both explicit and implicit semantic contexts. Our work includes an extensive experimental evaluation of the effectiveness and efficiency of our framework over three real-world datasets, case studies based on real user-generated examples and abstract intents, and comparison with the state-of-the-art in query reverse engineering (a special case of query intent discovery) and with PU-learning. Our empirical results highlight the flexibility of our method, as it is extremely effective in a broad range of scenarios. Notably, even though SQU\textsuperscript{ID} targets query intent discovery with a small set of examples, it outperforms the state-of-the-art in query reverse engineering in most cases, and is superior to learning techniques.

There are several possible improvements and research directions that can stem from our work, including smarter semantic context inference using log data, example recommendation to increase sample diversity and improve abduction, techniques for adjusting the depth of association discovery, on-the-fly αDB construction, and efficient αDB maintenance for dynamic datasets.

Acknowledgements: This material is based upon work supported by the NSF under grants CCF-1763423 and IIS-1453543. We thank Quoc Trung Tran for sharing the source code of TALOS.
10. REFERENCES


A. Srinivasan. The aleph manual.


