DICE: Data Discovery by Example

El Kindi Rezig
MIT CSAIL
elkindi@csail.mit.edu

Anshul Bhandari
NIT Hamirpur
185529@nith.ac.in

Anna Fariha
University of Massachusetts Amherst
afariha@cs.umass.edu

Benjamin Price
MIT Lincoln Laboratory
ben.price@ll.mit.edu

Allan Vanterpool
United States Air Force
allan.vanterpool@us.af.mil

Vijay Gadepally
MIT Lincoln Laboratory
vijayg@ll.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

ABSTRACT

In order to conduct analytical tasks, data scientists often need to find relevant data from an avalanche of sources (e.g., data lakes, large organizational databases). This effort is typically made in an ad hoc, non-systematic manner, which makes it a daunting endeavour. Current data discovery systems typically require the users to find relevant tables manually, usually by issuing multiple queries (e.g., using SQL). However, expressing such queries is nontrivial, as it requires knowledge of the underlying structure (schema) of the data organization in advance. This issue is further exacerbated when data resides in data lakes, where there is no predefined schema that data must conform to. On the other hand, data scientists can often come up with a few example records of interest quickly. Motivated by this observation, we developed DICE—a human-in-the-loop system for Data discovery by Example—that takes user-provided example records as input and returns more records that satisfy the user intent. DICE’s key idea is to synthesize a SQL query that captures the user intent, specified via examples. To this end, DICE follows a three-step process: (1) DICE first discovers a few candidate queries by finding join paths across tables within the data lake. (2) Then DICE consults with the user for validation by presenting a few records to them, and, thus, eliminating spurious queries. (3) Based on the user feedback, DICE refines the search and repeats the process until the user is satisfied with the results. We will demonstrate how DICE can help in data discovery through an interactive, example-based interaction.

1 INTRODUCTION

Data preparation is becoming the behemoth of data analytics pipelines [4, 6, 11]. The precursor of any data analytics task is to quickly find and link relevant data of interest from multiple sources such as enterprise databases and data lakes. Unfortunately, this puts a significant burden on data scientists because (1) querying the data requires knowledge of the underlying schema; and (2) linking multiple tables from different sources (e.g., data lakes) requires finding and assessing multiple possible join paths.

Through our collaborations with multiple partners, including the U.S. Air Force, we have observed that (1) data that are relevant for a specific data discovery intent are rarely contained within a small set of tables, but, rather, are spread across multiple tables from heterogeneous sources (e.g., data lakes); (2) users are often unaware of the underlying structure of those (typically heterogeneous) sources; and (3) users can often provide a few example records that represent the data they want to discover (e.g., if a user wants to retrieve all songs by Pink Floyd, they can provide the artist’s name and a song title as examples). Based on these observations, we developed DICE [12], an interactive data-discovery-by-example system, which can assist users in their data discovery tasks over data lakes.

Example 1.1 (U.S. Air Force). An organization within the Air Force is in charge of collecting data from dozens of sensor platforms to support data scientists in producing data-driven reports for decision-makers. The vast and heterogeneous data is organized across hundreds of tables in a data lake. Each table has a different schema that may be governed by sensor type or proprietary data handlers. While the end-user may have an idea of the type of records they wish to find, navigating the untamed data lake poses a major bottleneck in their data analytics pipelines.

Figure 1 shows example tables over the music domain from three separate sources on the right canvas (different colors denote different data sources). In a nutshell, DICE works as follows: In step 1, the user provides a few example records, without necessarily including the column names (top left table in Figure 1). Based on the examples, in step 2, DICE automatically finds join paths—paths that connect tables through Primary Key - Foreign Key (PK-FK) relationships (inferred from the data)—and selection predicates to construct “satisfying” SQL queries. Intuitively, a query “satisfies” a set of examples if its output includes the examples as well as other records that are similar to the examples (right canvas in Figure 1). Since multiple queries may satisfy the examples, we need a mechanism to figure out the correct query. To this end, in step 3, DICE presents to the user a small subset of diverse records from the output of one of the satisfying queries, and solicits their feedback to help it prune spurious queries. The user then approves or rejects each record (bottom left table in Figure 1). Based on the user’s feedback, DICE repeats steps 2 and 3 to explore alternative queries, until the user is satisfied with the final results.

Related work. Query by example (QBE) [7, 10, 13] is closely related to DICE as it also focuses on discovering SQL queries from user examples. However, existing QBE approaches make a strong assumption that data is stored in a well-defined schema where key-foreign-key relationships are known apriori. Some approaches [9, 15, 17] require a small database along with the corresponding example...
records as input; but, this requires complete schema knowledge. Beyond examples, a recent work considers natural language as an alternative means for specifying user intent [2]. Prior work on interactive data exploration [3, 5] shares some similarities with DICE, but they make simplified assumptions such as data resides in a denormalized table or support only a limited class of queries. In summary, none of the existing approaches are suitable for discovering data from data lakes—where no knowledge of the data organization is known in advance—while supporting expressive class of queries (selection, projection, and join) which DICE supports.

2 SYSTEM OVERVIEW

In this section, we present an overview of the building blocks of DICE (Figure 2). At first, the user provides a few example records (column names are not required). Then, DICE performs a fuzzy matching (i.e., similarity search) between the example values and the available data sources to extract matching columns. DICE then finds PK-FK relationships among the tables with matching columns. Finally, DICE shows a small set of records to the user for validation.

2.1 Example records

Format. The user provides a few example records, with named or nameless columns. If the column names are available, DICE tries to use those to prune the matching results later. DICE supports the following input value formats:

- **Text value:** If the user knows the values they are looking for, or part thereof, they can enter those directly into an example record. For ease of use, DICE features real-time keyword phrase suggestions as the user types some text.
- **Numerical value:** For numerical values, the user enters a number and DICE attempts to find the entered number.

Figure 1: Example DICE workflow: after obtaining the examples from the user in step 1, DICE finds join paths (arrows) across tables to construct output tables for the user to validate in step 2. The user provides feedback by either accepting or rejecting a record in step 3.

![Diagram of DICE workflow](image)

Figure 2: DICE has an interactive workflow to discover new data from user-provided examples. It starts by looking up the examples in the data sources, then it finds join paths, presents a few records to the user for validation, and repeats until the user is satisfied.

- **Range:** The user can also express numerical ranges to specify values that lie within specific ranges (e.g., 90 - 100).
- **Regular expression:** Additionally, the user can enter regular expressions if they know their intended format (e.g., dates).

**Example semantics.** Given the example records, DICE supports two data discovery semantics:

- **Keep Columns (KC):** In this case, the user wants to re-generate the example columns they provided and, in addition, enrich those columns with new values.
- **Keep Columns and Extend (KCE):** Similar to KC, this mode adds new values to the example columns, but, in addition, suggests new relevant columns (and values).
2.2 Looking up example values

In this phase, DICE looks up the example values in the available source tables. We pre-compute the min-hash of every source column to make the lookup efficient. Few questions arise here: should we search for all of the example column values at once (which could significantly limit the scope of the results), or should we select just a subset of the example columns, and, if the latter, which subset? For example, in the example of Figure 1, we can search using values in col1 only, which will require looking for all possible columns associated with Pink Floyd (e.g., years active, number of albums, etc.). On the other hand, if we consider col1 and col3, we will only get tracks (songs) whose user ratings are between 90 and 100.

We want to strike a balance between generating some of the values provided as examples and finding new values and columns of interest (which may not be part of the input example). In other words, we want to prevent both over-fitting and over-generalization. We adopt two search strategies depending on whether the user has knowledge of the column names in the example table or not:

Named columns. In this case, DICE attempts to use the provided column names to prune the space of matching columns in the source tables. DICE performs a keyword search of the provided column names (e.g., “song”) to find matching columns. Once column names are resolved, DICE expedites the lookup of the example values in the data sources as follows: During the \( i \)th iteration, it considers a subset of the example columns with cardinality \( i \), and incrementally keeps augmenting this subset in the subsequent iterations. For instance, in the example of Figure 1, assuming DICE knows that col1 corresponds to lyrics.artist and col2 corresponds to lyrics.name, DICE searches by the column lyrics.artist, and populates the column lyrics.track_name with all the tracks by Pink Floyd. This avoids the need to search by col1 and col2 together to find all Pink Floyd tracks, which significantly reduces the overhead for value matching and join path discovery.

Nameless columns. In many cases, especially in data lakes, column names are not meaningful; so the user might fail to provide the correct column names. In this case, DICE resolves the column mapping as follows: (1) it finds columns (and corresponding tables) from the data sources that match the example columns (based on their min-hash signatures); (2) it computes similarity profiles—modeled using min-hash signatures, following the procedure described in [8]—across all the columns of the tables found, and ranks these columns according to their similarity with the example columns; and (3) it attempts to join the tables with columns that contain similar values.

2.3 Finding join paths

Once DICE has identified the columns, the next step is to construct a SQL query that generates a table that includes the example records. To do so, we need to find the PK-FK relationships among the tables. While some data sources may have PK-FK relationships explicitly defined (e.g., normalized databases), many data sources today come from data lakes which do not have PK-FK relationships pre-defined. DICE automatically finds join paths (1) within a single source (e.g., data lakes) as well as across different sources (e.g., normalized databases and data lakes). We have found that this hybrid setting is the most realistic one since data scientists often have to sift through a mix of data sources (data lakes, enterprise databases, data warehouses, etc.) to conduct their analytical tasks (e.g., linking together a marketing database, a sales database, and a global company data lake to extract informative product sales features).

Based on the similarity profiles computed in the lookup phase, multiple possible join paths may exist to join two or more tables (e.g., in Figure 1, column artist in table ratings is similar to column name in table artist). DICE joins tables whose columns are most similar to each other, and, thus, indicate a potential PK-FK relationship. DICE strives to conserve coverage of all the example column values because they fall within the user’s interest. During each iteration, DICE generates at most \( n \) join paths (\( n \) is user-provided, 3 by default) that cover all the example values. DICE generates the next \( n \) candidate join paths in the subsequent iterations until the user is satisfied with the results.

2.4 Building and pruning example records

From the selected join path, DICE generates records for user validation. Since the number of records that result from a join path can be very large, DICE shows \( k \) (\( k \) is user-specified, 20 by default) records to the user for validation and strives to (1) present a diverse sample of records in terms of values; and (2) include the user-provided example values. Since there is a trade-off between value diversity and coverage of the example records, DICE allows the user to specify a coverage threshold (e.g., 80%) that indicates what fraction of the example records must be included in the results from a join path.

3 DEMONSTRATION SCENARIO

The data from the Air Force is not public, so we will demonstrate DICE over the following public datasets in the music domain as this data domain is of universal interest:

- MusicBrainz [14] contains over 200 tables that include detailed information about musical artists, their records (songs), releases (albums), production dates, etc.
- MusicXMatch [1] was crawled from AZLyrics and contains about 150K song titles with their lyrics.
- Yahoo! user ratings [16] contains user ratings of over ten million song titles.

Demonstration outline. DICE allows user interaction, where at each step, the user can provide feedback to refine the results. Through our demonstration, we aim to (1) show the participants various steps of DICE; (2) allow them to engage with DICE by entering their own example records and by walking through the data discovery steps; and (3) allow the participants to interact with DICE through multiple iterations where they will validate records during the feedback solicitation phase. Figure 3 illustrates the DICE interface built within a Jupyter Notebook. We chose Jupyter notebook as it is extremely popular for developing data-analytics pipelines, and data scientists are usually familiar with its interface. We describe the demonstration scenario through the following steps:

1. **Providing example records**: In the first step (top left canvas of Figure 3), the user enters example records (e.g., artist names, song titles, etc.). To facilitate this step, DICE provides keyword phrase suggestions as the user types example values.
**Figure 3:** The DICE interface: the four canvases represent points of interaction with the user during the data discovery process.

**1.** Excluding irrelevant tables: To aid the user in narrowing down the relevant tables, DICE shows a snippet that displays the tables that it is currently considering to generate the PK-FK graphs (bottom left canvas in Figure 3). The goal here is to allow the user to discard tables that they deem irrelevant to their search, which will lead to fewer iterations to reach a satisfactory result.

**2.** Enumerating candidate join paths: DICE displays all join paths being considered to construct the candidate queries and highlights the ones that are being expanded in the current iteration (the red arrow in the bottom right canvas in Figure 3). DICE allows the user to override the pre-selected paths by selecting different ones.

**3.** Generating example records and feedback solicitation: After DICE expands the chosen join paths, it presents a few records to the user for review (top right canvas in Figure 3). The user then examines the records and provides feedback for each record by either accepting or rejecting it, which helps DICE refine the search in the later iterations.

In the subsequent iterations, the user can enter more examples, or simply let DICE explore alternative join paths and repeat the four steps until they are satisfied with the final results.

**Demonstration engagement.** After our guided demonstration, participants will also be able to use DICE to explore other real-world datasets (e.g., IMDB, Box Office Mojo). As DICE can work on any data domain, particularly since DICE was designed for Air Force and Navy use cases, participants will be able to plug their own datasets into DICE.

Through the demonstration, we will showcase how DICE can effectively discover data that are of user’s interest based on a few examples and interactions. The key takeaway is that interactive and example-based data discovery aids in data retrieval from data lakes, where no predefined schema exists.

**REFERENCES**


