Data-Driven Insight Synthesis for Multi-Dimensional Data

Junjie Xing
University of Michigan
Ann Arbor, Michigan, USA
jjxing@umich.edu

Xinyu Wang
University of Michigan
Ann Arbor, Michigan, USA
xwangsd@umich.edu

H. V. Jagadish
University of Michigan
Ann Arbor, Michigan, USA
jag@umich.edu

ABSTRACT

Exploratory data analysis can uncover interesting data insights from data. Current methods utilize "interestingness measures" designed based on system designers' perspectives, thus inherently restricting the insights to their defined scope. These systems, consequently, may not adequately represent a broader range of user interests. Furthermore, most existing approaches that formulate "interestingness measure" are rule-based, which makes them inevitably brittle and often requires holistic re-design when new user needs are discovered.

This paper presents a data-driven technique for deriving an "interestingness measure" that learns from annotated data. We further develop an innovative annotation algorithm that significantly reduces the annotation cost, and an insight synthesis algorithm based on the Markov Chain Monte Carlo method for efficient discovery of interesting insights. We consolidate these ideas into a system. Our experimental outcomes and user studies demonstrate that DAISY can effectively discover a broad range of interesting insights, thereby substantially advancing the current state-of-the-art.

PVLDB Reference Format:
Junjie Xing, Xinyu Wang, and H. V. Jagadish. Data-Driven Insight Synthesis for Multi-Dimensional Data. PVLDB, 17(5): 1007 - 1019, 2024.
doi:10.14778/3641204.3641211

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/GavinXing/DAISY_VLDB.git.

1 INTRODUCTION

Exploratory data analysis has become increasingly crucial in data analysis and decision-making across different domains [9, 28, 45], such as business intelligence. Recent Gartner Reviews [5, 8] have highlighted it as an emerging topic that could automatically discover and surface important insights. As a result, the generation of interesting and useful insights has become an important problem, gathering significant attention in recent research [12, 31, 43].

Insights are results of queries executed against the database that expose interesting patterns in the data. An infinite number of queries can be posed against any given database, and only a small number of these generate interesting insights. Since this is a problem of importance, several previous papers have developed techniques to address it. The typical solution comprises three parts. ***First***, identify the types of insights to focus on. For example, QuickInsights [12] defines 13 types of insights, such as trend, outlier, majority, etc. MetaInsight [31] defines two types of insights, commonness and exception, based on homogeneous data patterns. ***Second***, formulate the interestingness measure for each type of insights, which evaluates the level of interestingness an insight presents. QuickInsights, for instance, defines two major components, "impact" and "significance", and formulates them for each insight type based on their observations. ***Third***, optimize the mining process. Researchers develop different techniques to speed up the mining process based on the characteristic of the insight types and the interestingness measure. For instance, QuickInsights leverages two aspects of its interestingness measure (a) "impact" is less computationally intensive than "significance" (b) the interestingness measure is proportional to "impact", and develop several optimization rules. In this work, we develop a new paradigm for building interestingness measure by learning from labeled data (as depicted in Figure 1).

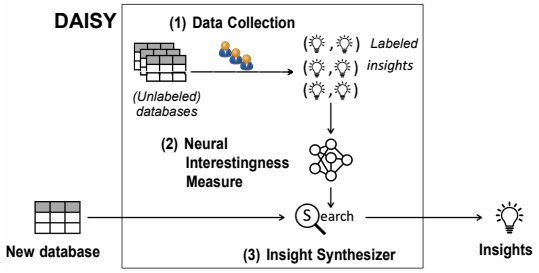


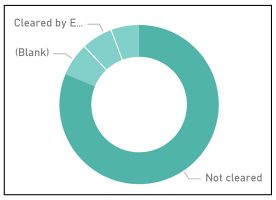
Figure 1: Schematic workflow of DAISY.

Motivation. The formulation of the interestingness measure plays a vital role in insight discovery, determining the insights a system can render. While *hand-crafted* interestingness measures are effective within their specific problem domains, their interestingness scope is limited by the characteristics defined by the developers. This could lead to users with different interests missing many interesting patterns due to the developers' oversight or underestimation of these patterns' significance.

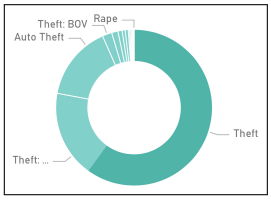
To validate this claim, we conducted a large-scale experiment with the Austin Crime database [1], where users were asked to identify interesting data patterns. QuickInsights, as shown in Figure 2, is able to discover some insights that are deemed interesting according to our user study. However, the insights discovered by QuickInsights are restricted to data patterns that the system designers could think of. As a result, it also missed some other important insights indicated by users, such as the two shown in Figure 3.

This experiment demonstrated that while predefined *rule-based* interestingness measures can surface insights *with high precision*,

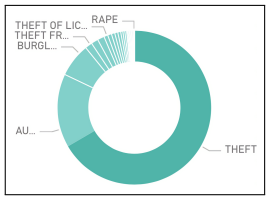
Proceedings of the VLDB Endowment, Vol. 17, No. 5 ISSN 2150-8097.
doi:10.14778/3641204.3641211



(a) More than 50% of the cases are "not cleared".

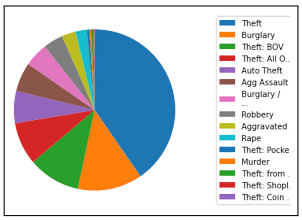


(b) More than 50% of the cases are of "theft" type.

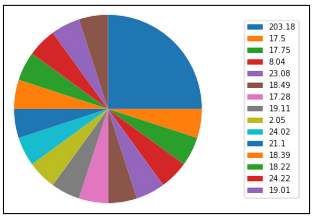


(c) More than 50% of the cases are described as "theft".

Figure 2: Top 3 Insights generated by QuickInsights for the Austin Crime database. The highest ranked insight generated by QuickInsights (see Figure 2a) states that a majority of the incident records are not cleared. This insight is essentially generated by first executing a SQL query that counts the number of records for each clearance status and then calculates the percentage of each status. QuickInsights discovered some other interesting insights as well; for instance, Figure 2b and Figure 2c show two insights that are ranked second and third. As we can see, these insights in Figure 2 are very similar to each other. In fact, they all belong to the same insight type in QuickInsights, called "attribution": this type considers only insights where the leading value in one column dominates *more than 50%* of values in the same column [12].



(a) For census tract 18.04, top 4 crime types combined account for nearly 80% of all crimes in census tract 18.04.



(b) For clearance date 2015-12-06, top 3 census tracts combined account for about 35% of all crimes in Austin.

Figure 3: Visualizations of two insights that are ranked very high in our user study. DAISY is able to discover both of them; however, QuickInsights cannot. The insight shown in Figure 3a reveals a data pattern that, for crimes in the census tract 18.04, the *top 4 crime types combined* accounts for nearly 80% of all crimes conducted in that area. This insight was ranked very high according to our survey; however, QuickInsights cannot generate it because it has a rule (which says, "the leading value should account for more than 50% of total value") that is required to hold *for any interesting insight*, but it does not hold for this particular pattern. Similarly, Figure 3b shows another example insight that was ranked highly in our survey and that can be successfully discovered by DAISY but not by QuickInsights.

they fundamentally lack the capacity to *cover* a diverse array of data patterns that are *broadly interesting*.

Our Approach. Motivated by the findings, we propose *data-driven* insight synthesis, enabling the system to learn an interestingness measure *from data*, guiding the subsequent insight search process.

This is realized through a new data-driven insight synthesis system, DAISY[1], whose workflow is schematically shown in Figure 1. To use our system, a user simply uploads a multidimensional database; DAISY then generates interesting insights from it. Internally, DAISY uses a pre-trained interestingness measure that quantitatively scores candidate insights[2] in the provided database. Given our interestingness measure, DAISY then searches for interesting insights, which are finally visualized to users.

Challenges. To derive a *data-driven* interestingness measure from user annotation data, and to employ DAISY with the derived interestingness measure, we need to address the following challenges.

Labeling insights efficiently. The first challenge is the efficient collection of *labeled* insights across a diverse range of databases. Even if we restrict ourselves to one database and insights corresponding to single-block SQL queries, millions of insight candidates can still be generated from these queries. It is infeasible to exhaustively rate all these insights and form a total order based on their interestingness. In this work, we propose a methodology that requires annotating significantly fewer insights while still being able to approximately rank them effectively. The core idea is to collect *partially-ordered* insights. We also propose a dynamic annotation algorithm that focuses on labeling more interesting insights based on the current annotation results. More details about our data collection methodology are described in Section 2.

Training an effective interestingness measure. The second challenge is using the collected data to train an effective interestingness measure. To achieve this, we employ neural networks to approximate an interestingness measure. Our neural network model assigns higher scores to insights ranked higher in user annotations. The details can be found in Section 3.

Searching for interesting insights. Finally, the third challenge is developing an insight synthesizer that can effectively search for useful insights in a *new* database using the neural interestingness measure. Different from prior work [12, 31] where the interestingness measure is human-crafted explicit rules, which can be used to accelerate the mining process. Our neural model has a less clear structure, making the search significantly more challenging. We propose a search algorithm agnostic to the interestingness measure's underlying structure to address this challenge. In particular, we formulate the problem of insight search as an optimal program synthesis problem and develop a synthesis algorithm based on Markov Chain Monte Carlo; more details can be found in Section 4.

[1] DAISY represents DAta-driven Insight SYnthesis.

[2] For brevity, we refer to candidate insights as simply "insights" in the rest of this paper

We have implemented DAISY (Section 5) and evaluated it through a series of experiments (Section 6). We discuss related work in Secton 7, and give our conclusion in Section 8.

In summary, this paper makes the following contributions:

- We introduce a *data-driven* interestingness measure, learnt from user annotation data, and propose an effective and efficient method for collecting training data.
- We define the insight search problem as an optimal program synthesis problem and develop an efficient insight synthesis algorithm using Markov Chain Monte Carlo.
- We implement our ideas in a new system, and evaluate its effectiveness using ten real-world databases and a user study.

2 LABELING INSIGHTS

Our objective is an interestingness measure that captures a broader spectrum of interestingness with user annotation. So we have to develop training data with care. We begin by constructing a new training dataset from 10 diverse real-world databases (details of these databases can be found in Section 5.1).

Recall that our interestingness measure model takes in an insight (table) and returns a numerical value indicating its interestingness. Therefore, a straightforward way to train such a model is to develop a corpus of insights on this dataset where every insight is labeled with its interestingness score; this essentially gives us a *total order* of insights. Unfortunately, this idea is not feasible because constructing a total order of insights requires an enormous amount of manual annotation effort, and it is very difficult to scale this process to multiple large databases. Furthermore, it may not always be easy to create a total order: for instance, there may be multiple insights that are similar to each other in terms of interestingness, and humans may differ in their relative ordering. In other words, a unique total order in general may not even exist.

Our Observation. After investigating the set of insights that users found interesting in our preliminary experiment mentioned in the introduction, we come up with two observations: (1) many insights are similar with each other, (2) similar insights share similar interestingness level.

Key ideas. We propose to use *partially-ordered* insights as our training data. In particular, we create a corpus of *labeled insight pairs*, as opposed to labeling each insight with the absolute rank/score. Given a pair (I, I'), we consider three types of labels: (a) if the label is 1, it means I is more interesting than I', (b) if the label is -1, it means I is less interesting than I', and (c) if the label is 0, it means both insights are similar in terms of their interestingness. The advantage of this approach is that, rather than annotating a set of candidate insights to form a total order, we only need to label which insight (among two candidate insights) is more interesting. It does not require exhaustively labeling all insight pairs; we only need to obtain a set of labeled pairs that cover sufficiently many candidate insights from the given database. Finally, cognitive psychology researchers also show that providing pairwise comparisons is typically much more straightforward than quantitatively measuring each individual entity [14, 15, 17].

In what follows, we first describe how we collect data.

Generating candidate insights by enumerating queries. Given a multi-dimensional database, our first step is to generate all candidate insights. To do this, we enumerate *all* queries from the query language we consider and run each of them against the database. Details of the query language will be introduced in Section 4.1.

Sampling insight pairs using k-means clustering. Our next step is to generate *insight pairs* to be annotated. However, because there are too many of them (quadratic to the number of insights), it is infeasible to annotate all insight pairs. Therefore, we sample a subset of them to annotate. Our basic idea is to first cluster all insights into n clusters $C_1, \ldots, C_n$, then randomly sample an insight I_i from each cluster C_i, and finally obtain a set of insight pairs from these sampled insights.[3] That is, the final set R of insight pairs is:

$$R = \{(I, I') \mid I \in C_i, I' \in C_j, 1 \leq i < j \leq n\}$$

Our technique uses the k-means algorithm to partition all insights into clusters. In particular, we first compute a feature vector for each insight I: this feature vector contains the normalized top-10 values of the aggregated column of I. Then, based on these feature vectors, the k-means algorithm returns n disjoint clusters $C_1, \ldots, C_n$. These clusters are used to construct the final set R of insight pairs.

Example 2.1. Figure 3 presents two insights. The aggregated column, the number of crimes by different crime types and different census tracts, is used in each to calculate their feature vectors for clustering. Given the disparity of the two insights, we expect them to reside in different clusters. The details of the insight formulation and query language will be discussed in Section 4.1.

Annotating insight pairs dynamically. Unfortunately, even if we restrict to only those sampled insight pairs, annotating all of them still incurs prohibitive overhead (since $|R|$ is quadratic to n). For instance, if we have 10^3 clusters, we will have 10^6 insight pairs to annotate. To further reduce this burden, our key observation is that it is not necessary to *exhaustively* compare the interestingness for those *less interesting* insights. That is, it is less useful for R to include insight pairs (I, I') where *neither of* I, I' are *interesting*. The reason is that our ML model only needs to score those interesting insights precisely–this is important for us to identify top-ranked insights; it does *not* need to precisely differentiate the relative interestingness of the less interesting insights, as long as their scores are all lower than the most interesting ones. Therefore, motivated by this idea, we only exhaustively construct pairs for the most interesting insights.

However, there is a cyclic dependency underlying this idea: how do we know an insight is interesting before its interestingness is annotated? To break this cyclic dependency, we propose an algorithm that dynamically determines which insights to annotate based on the current interestingness annotations. Instead of having two separate steps where we first construct all insight pairs and then annotate them, we intertwine these two steps using a quickselect-like algorithm. In particular, given a set of n insights, the goal of our algorithm is to produce

$$[I_1, \ldots, I_k], \{I_{k+1}, \ldots, I_n\}$$

such that: (a) for any $1 \leq i < j \leq k$, I_i is annotated to be more interesting than I_j, and (b) for any $1 \leq i \leq k < j \leq n$, I_i is annotated to be more interesting than I_j. In other words, our algorithm

[3] In our implementation, we actually sample multiple insights from each cluster. More details can be found in Section 5.

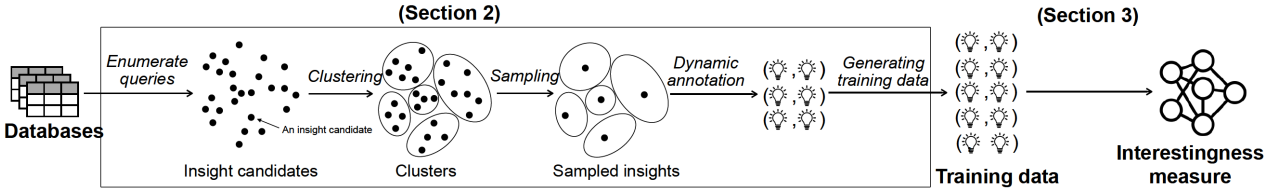


Figure 4: Schematic workflow of our neural interestingness measure training process.

constructs a total order of the top-k insights but it does not care about the relative interestingness for insights $I_{k+1}, \ldots, I_n$.

More specifically, our algorithm is based on the standard quickselect algorithm for finding the kth largest element in an unsorted list. In particular, given n insights, it first picks a pivot insight I_p, making sure all insights that are more interesting than I_p are placed to the left of I_p and all less interesting ones are placed to its right. Note that we need humans to annotate their relative interestingness when comparing two insights in this algorithm. If I_p is among the top-k insights[4], we recursively invoke our algorithm for insights to the right of I_p. Otherwise, we invoke the algorithm for those to the left. This recursive process terminates until we obtain a pivot insight that is at exactly the kth position.

Note that, at this point, we have also identified the top-k insights. However, their relative interestingness is not yet determined. Therefore, we use a standard sorting algorithm (e.g., quicksort) to sort the top-k insights, where every comparison of interestingness requires human annotations. We also note that we do not sort the remaining *low-ranked* insights, because it is not necessary to precisely determinate their relative interestingness.

Generating training data from annotated insight pairs. In the final step, given the top-k sorted insights $[I_1, \ldots, I_k]$ and m lower-ranked unsorted insights $\{I'_1, \ldots, I'_m\}$, we create a set S_1 of directly-labeled insight pairs as follows. For each label (I, I', Z), I and I' are insights and $Z \in \{-1, 0, 1\}$ is the label. Specifically, given insights I and I', $Z = 1$ means I is *more* interesting than I', whereas $Z = -1$ encodes I is *less* interesting than I'. In addition, $Z = 0$ means I and I' are similar in terms of their interestingness.

$$S_1 = \{(I_i, I_j, 1) \mid 1 \le i < j \le k\} \cup \{(I_i, I'_j, 1) \mid 1 \le i \le k, 1 \le j \le m\} \quad (1)$$

In addition, given the clusters $C_1, \ldots, C_n$ obtained from k-means clustering, we create the following two sets of labeled insight pairs.

$$S_2 = \{(I_i, I_j, Z) \mid (I, I', Z) \in S_1, I \in C_i, I' \in C_j, I_i \in C_i, I_j \in C_j\} \quad (2)$$

$$S_3 = \{(I, I', 0) \mid I \in C_i, I' \in C_i, 1 \le i < j \le n\} \quad (3)$$

The intuition is that insights from the same cluster are "similar". Therefore, if in S_1, I is more interesting than I', we can infer that any insight I_i in I's cluster C_i is more interesting than any insight I_j from the cluster C_j of I'. For S_3, it simply says any two insights I, I' from the same cluster are similar to each other.

Our final dataset S is the union of S_1, S_2, S_3, i.e., $S = S_1 \cup S_2 \cup S_3$. This significantly boosts the amount of our training data.

Limitations. We outline two key limitations of the labeling process. First, the labeling process necessitates the involvement of a developer with expertise in clustering techniques and crowd-sourcing. This individual should have domain-specific knowledge and a deep understanding of the datasets under consideration, enabling them to configure clustering parameters effectively. While this requirement is not inherently limiting, we anticipate that DAISY's developers would meet the requirements. Second, the quality of the training set is tied to the effectiveness of the clustering algorithm. In our experiments, we empirically determined the hyper-parameter by exploring various cluster settings. The ultimate quality of the training data hinges on the success of this clustering process.

[4]Here, k is a hyperparameter of our annotation algorithm.

3 INTERESTINGNESS MEASURE FROM DATA

In this section, we first give our problem formulation for interestingness measure from data in Section 3.1, then we introduce the neural model and the training method in Section 3.2.

3.1 Problem Formulation

We define an insight as a *table*, which is a query result that may exhibit an interesting data pattern. Under this definition, our interestingness measure is a function that takes as input an insight I (i.e., a table) and returns a numerical score indicating how interesting I is; this is fairly standard in the literature [12]. However, unlike prior work that uses manually-crafted rule-based measures, we aim to learn this measure from data automatically.

Ideally, the training data is a set of candidate insights, each annotated with an interestingness score. However, as discussed in Section 2, such annotations are difficult to obtain, primarily because it requires users to assign scores to insights with no handy absolute scale that users can refer to. Instead, we propose to label *partially-ordered* insights instead of a total order. Given the training data, we formulate the learning problem as follows.

Definition 3.1 (Interestingness measure over data)**.** Given a set S of *labeled insight pairs* of the form (I, I', Z), where I and I' are insights and $Z \in \{-1, 0, 1\}$ is the label. Specifically, given insights I and I', $Z = 1$ means I is *more* interesting than I', whereas $Z = -1$ encodes I is *less* interesting than I'. In addition, $Z = 0$ means I and I' are similar in terms of their interestingness. We define an *interesting measure* M as a function that takes as input an insight I and returns a positive numerical value. Furthermore, M should minimize the following loss L:

$$\sum_{(I,I',Z)\in S} (1-|Z|)\cdot|M(I)-M(I')|+|Z|\cdot\max\{0, m-Z\cdot(M(I)-M(I'))\}$$

Here, m is the margin, which is a hyper-parameter to our problem. When label $Z = 0$, i.e. I, I' are similar in terms of interestingness, the loss is $|M(I) - M(I')|$, minimizing the loss will make the model learn to assign close interestingness scores to the two insights;

primary_type	COUNT(*)
Theft	520
Burglary	167
Theft: BOV	135
Theft: All Other Larceny	110
...	...

```
SELECT      primary_type, COUNT(*)
FROM        Austin Crime
WHERE       census_tract="18.04"
GROUP BY    primary_type
ORDER BY    COUNT(*) DESC
```

(a) Insight (left) and its corresponding query (right) for Figure 3a.

census_tract	COUNT(*)
203.18	5
17.5	1
17.75	1
8.04	1
...	...

```
SELECT      census_tract, COUNT(*)
FROM        Austin Crime
WHERE       clearance_date ="2015-12-06"
GROUP BY    census_tract
ORDER BY    COUNT(*) DESC
```

(b) Insight (left) and its corresponding query (right) for Figure 3b.

Figure 5: Two example insights.

when label $Z = 1$, i.e. I is more interesting than I', the loss is calculated as $\max\{0, m - (M(I) - M(I'))\}$, minimizing the loss will ask the model to learn to assign a higher interestingness score to I than I' with a margin of m; the same logic applies when $Z = -1$.

Example 3.1. The insight/table in Figure 5a corresponds to the visualization from Figure 3a; this insight is obtained by executing the SQL query (on the right) which considers crime records in a particular census tract (namely, 18.04). This insight is interesting since multiple crime types *combined* account for a vast majority of all crimes for the given area. Similarly, the insight in Figure 5b is also interesting for a similar reason.

3.2 Interestingness Measure Model

After collecting the training data, the next step is to use it to train a model that can predict an interestingness measure for any candidate insight. Recall that our Definition 3.1 of interestingness measure is written as the minimization of a loss function. A multilayer perceptron (MLP) model is usually a good choice of neural architecture [10] to solve such an optimization problem.

Multilayer Perceptron Interestingness Measure. Neural network models have demonstrated remarkable success in various fields. For instance, recurrent neural networks, short term memory models and transformer-based models have gathered significant attention in natural language processing tasks. While these models have excelled with long sequences, particularly in the context of text inputs, our unique problem of measuring interestingness over data requires a distinct approach. In our case, we have deliberately chosen the Multilayer Perceptron (MLP) for several reasons. First, MLP represents a simpler yet well-established neural architecture when compared to the aforementioned models. Its maturity is underscored by a rich literature encompassing methodologies, efficient training techniques, and refinements. As such, MLP serves as an excellent baseline model within the DAISY system. Moreover, MLP possesses the valuable characteristic of being a universal approximator, adepting at capturing nonlinear relationships, which is a crucial feature in modeling our interestingness measure [10]. This capability aligns with the demands of our problem domain and adds to MLP's suitability as the model of choice.

The crucial next question is what features to consider. In the next paragraph, we describe our choices. However, we should first hasten to point out that the intellectual contributions of this paper are orthogonal to the neural model architecture or features being chosen. If readers would like to use other neural models or features in a system they build, they can freely adopt their choice while keeping everything else described in this paper intact. The key novelty of our work lies in that we apply this neural model in a new problem domain for scoring insights, along with an efficient training data collection method, discussed in Section 2.

Features. We choose to manually engineer features instead of using end-to-end deep representation learning, which is not very suitable for our problem of insight ranking, where the insights/tables involve structured data with a lot of numbers [6]. Given an insight I (i.e., a table), we first extract features from the entire table I (e.g., the number of tuples). In addition, we also extract features from the aggregated column C in I (which is the column resulted from an aggregation function in the corresponding query that produced I). On the other hand, we do not use dimension columns, since in our experience, most of the interesting insights are related to values in the aggregated column. More specifically, we consider the following features:

(1) *Tuple-count*: the total number of tuples in I.
(2) *Value-sum*: the sum of all values in C.
(3) *Max-min*: the maximum and minimum values in C.
(4) *S-dev*: the standard deviation of all values in C.
(5) *Top-10*: the percentage of each of the top-10 values in C.

We choose the features from two perspectives. *First*, feature (1-4) encompass statistical metrics, they are also employed in previous work [30]. These statistical features provide a comprehensive statistical summary of the insight. *Second*, feature (5) the raw data (normalized) representing the insight itself. We use the raw data and subsequently truncate the "top-10" to form a fixed length input for our neural model. This selection decision stems from the absence of an effective encoder for numerical table columns.

4 INTERESTING INSIGHT SYNTHESIS

So far, we have described how to train a machine learning model that can be used to measure the interestingness of an insight. In this section, we present a synthesis algorithm that can automatically generate interesting insights from a multi-dimensional database given an interestingness measure. Our synthesis algorithm is parameterized over an interestingness measure and, therefore, can be used in combination with other interestingness measures as well.

Key idea. We view insight mining as an optimal program synthesis problem. Different from prior work on both program synthesis [2, 22, 34] and insight mining [12, 31], a unique challenge in our work is that the interestingness measure is a neural network. As a result, there is no explicit structure from the interestingness measure that we can leverage to guide the search. To address this challenge, we propose using a *stochastic search* method in this work. In particular, our approach is based on Markov Chain Monte Carlo (MCMC) sampling, a well-known technique that can effectively solve optimization problems with poor structures.

In what follows, we first give our problem formulation for insight synthesis, then we give a primer on MCMC. Last, we present our MCMC-based insight synthesis algorithm.

4.1 Insight Synthesis as Program Synthesis

Recall that insights are defined as tables obtained by executing SQL queries against data. Therefore, we can reduce the problem of generating interesting insights to the problem of synthesizing "interesting" SQL queries that produce these insights; here, the interestingness of a SQL query is defined by the interestingness of its resulting table. Now, we can view the insight synthesis problem as an optimal program synthesis problem with *a semantic objective*. That is, we aim to synthesize a program (i.e., a SQL query) whose output (i.e., the insight) maximizes a quantitative objective (i.e., the interestingness measure). In what follows, we first give a primer on optimal program synthesis. Then, we formalize our query language. Finally, we present our definition of the insight synthesis problem.

4.1.1 Primer on Optimal Program Synthesis. Program synthesis, in general, is concerned with the problem of how to automatically generate a program from its desired specification. A standard framework for formulating program synthesis problems is syntax-guided synthesis (SyGuS) [2], where the key idea is to explicitly define a *space of programs* of interest using context-free grammars (CFGs). Then, a *search algorithm* will search for a program within this space that satisfies a given *specification* ϕ. This specification ϕ could take various forms such as input-output examples or logical constraints. In general, SyGuS is not restricted to particular forms of specifications or search algorithms, as long as there is a way to check whether a program satisfies the given specification.

Optimal program synthesis is a generalization of SyGuS where the task searching for a *highest-score* program P that satisfies the specification ϕ. That is, an optimal program synthesis problem also requires an additional scoring function $f : L \to \mathbb{R}$ that assigns a numerical score to each program P in the language L (i.e., the search space). The formal definition is given below.

Definition 4.1 (Optimal Syntax-Guided Program Synthesis)**.** Given a language L of programs, a scoring function $f : L \to \mathbb{R}$, and a specification ϕ, the optimal syntax-guided program synthesis problem is to find a program $P \in L$ such that P satisfies ϕ and $f(P)$ is maximized among all such programs in L satisfying ϕ.

4.1.2 Query Language. To formalize insight synthesis as an optimal SyGuS problem, we first define a context-free grammar (CFG), which essentially defines a search space of *candidate* queries that may produce interesting insights. The specific query language we illustrate in this paper is a subset of SQL and uses standard SQL operators, logical predicates, and aggregation functions, as shown in Figure 6. In particular, we focus on multi-dimensional data where tables consist of two types of columns: *dimension* columns (denoted DC) and *measure* columns (denoted MC). Our language considers queries of the same structure: they select some columns from an input table, filtered by a predicate φ; the resulting table is grouped by certain dimension columns and ordered by a particular measure column. Our language allows using aggregations, such as `SUM`, `AVG`, `COUNT`. Predicates in our language can refer to a string s.

The semantics of our examplar query language follows the standard SQL semantics. We use $[\![P]\!]_D$ to denote the evaluation of a query P on a database D; the evaluation result is a table/insight. Readers can refer to Example 3.1 for some example queries as well as their corresponding insights.

Generalization of Query Language. It is important to highlight that our technique is agnostic to the query language. The query language defined in Figure 6 is an instance of valid query languages for the explanation, experimentation, and evaluation purpose. System designers can define a query language for their specific domain. For example, the **SELECT** statement can include more measure columns, more operators can be used in the predicate ϕ, and nested query can also be allowed in the query language. As long as the query language and the corresponding transformations (to be introduced in Section 4.3) maintain the properties of the Markov chain (to be introduced in Section 4), it can be directly used by our system.

$$
\begin{array}{rrcl}
\textit{Query} & P & ::= & \textbf{SELECT}\ DC, \ldots, DC, \psi(MC) \\
 & & & \textbf{FROM}\ T \\
 & & & \textbf{WHERE}\ \varphi \\
 & & & \textbf{GROUP BY}\ DC, \ldots, DC \\
 & & & \textbf{ORDER BY}\ \psi(MC)\ \textbf{DESC} \\
\textit{Predicate} & \varphi & ::= & DC = s \mid \varphi \wedge \varphi \\
\textit{Aggregation} & \psi & ::= & \texttt{SUM} \mid \texttt{AVG} \mid \texttt{COUNT} \mid \ldots
\end{array}
$$

Figure 6: CFG of our query language. DC is a dimension column. MC is to a measure column. s is a string constant.

4.1.3 Insight synthesis. The problem definition is given below.

Definition 4.2 (Insight Synthesis Problem)**.** Given the query language L defined in Figure 6, an interestingness measure $M : \mathcal{I} \to \mathbb{R}$ that assigns numerical values to insights, and an input database D, the insight synthesis problem is to find a program $P \in L$ such that $M([\![P]\!]_D)$ is maximized among all programs in L.

While Definition 4.2 is based on Definition 4.1, it is distinct in two important ways. First, our definition uses a *semantic* objective M, rather than a syntactic objective (which is standard in program synthesis literature), because our objective requires *executing* the query P on the input database. This is necessary because interestingness is a property of query results rather than queries. A second distinction is that Definition 4.2 does not involve specifications (which constrain the functional behavior of queries). We are only interested in synthesizing a query that maximizes the quantitative objective. This is reasonable because it may not always be easy for users to provide such functional constraints.

4.2 Primer on Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods comprise a collection of sampling algorithms for drawing elements from a probability distribution. In particular, given a probability density function, it can sample elements from higher-probability regions more often than those from regions of low probability.

MCMC has been used in a variety of application domains, one of which is *score maximization*. That is, given a score function s that assigns a score to each element e in the search space, MCMC can be used to effectively search for elements with the highest score. The idea is to construct, from s, a probability density function p which can be directly used in MCMC to find a highest-score element. For instance, one commonly used p is the following [18].

$$p(e) = \frac{1}{Z} \exp\left(\beta \cdot s(e)\right)$$

Here, β is a constant and Z is a partition function that normalizes the distribution. While it is in general intractable to compute Z, the Metropolis-Hastings algorithm [20, 33] can be used to explore the density function without necessarily computing Z. It is a standard and well-known method to solve the MCMC sampling problem. In particular, at each step, the algorithm maintains a current element e. In the next step, it transforms e to another element e' (which is also called a *proposal*). This proposal e' can be accepted or rejected, which is controlled by : if accepted, e' becomes the current element; otherwise, another proposal is generated. This process is repeated a number of times (e.g., until it times out), and returns an element with the highest score among all elements searched so far.

4.3 Synthesizing Insights using MCMC

While MCMC appears promising at an intuitive level, actually using it requires casting the program synthesis problem into an MCMC framework. This is not straightforward because: (1) we need to ensure that under the insight synthesis problem setting, the Markov chain can converge (i.e., the *ergodic* property is maintained); (2) the *proposal distribution* should be appropriately designed w.r.t the insight synthesis problem and the *query language* of choice.

Now, let us describe how our insight synthesis algorithm works. At a high level, we use MCMC sampling to search for high-score programs according to our interestingness measure: each element in the search space is a SQL query, and we use our interestingness measure M as the score function. Given a query P in our language (from Figure 6), our proposal distribution $q(P \rightarrow \cdot)$ gives the probability distribution of transformed queries P'. We consider the following four types of transformations.

Replacing measure column. First, we consider transformations that replace MC with another measure column MC' in the same table, with probably p_m. In particular, we sample MC' among all measure columns uniformly at random. For instance, if there are ten measure columns, each will have a probability $p_m/10$.

Changing dimension column. Similarly, we also consider replacing one of the *dimension* columns in P with another dimension column in the same table, with probability p_d. We also sample the target dimension column uniformly at random.

Transforming aggregation. The aggregation function ψ used in P may also get transformed to another, with probability p_a. Again, we sample the target aggregation uniformly at random, among all aggregation functions available in our query language.

Rewriting filter. The final transformation rewrites filters. In particular, a filter $DC = s$ in P may get rewritten to another filter, with probability p_f. We consider all filters of the same shape but with different dimension columns or values from the same table. As before, we generate the new filter uniformly at random.

Note that we need to ensure $p_m+p_d+p_a+p_f = 1$. Furthermore, it is also pretty obvious that our transformations are ergodic, because any query in our query language can be transformed to any other query using a sequence of steps of the four types above.

Generalization of the Transformations It is essential to note that the above transformations are tied with the query language defined in Figure 6. If the system designer instantiates the system with another query language, they need to design the transformations accordingly so that the ergodic property is maintained.

```
SELECT    status, COUNT(*)
FROM      Austin Crime
WHERE     census_tract=18
GROUP BY  status
ORDER BY  COUNT(*) DESC
```

(a) Current query

```
SELECT    status, COUNT(*)
FROM      Austin Crime
WHERE     census_tract=20
GROUP BY  status
ORDER BY  COUNT(*) DESC
```

(b) Proposal query

Figure 7: An example that illustrates how query transformations work. Here, the proposal query in Figure 7b is obtained by rewriting the filter in the current query from Figure 7a.

5 IMPLEMENTATION

In this section, we describe some important implementation details for both the interestingness measure and the search technique.

5.1 Neural Interestingness Measure

We present how we sample insights, give more details about our annotation process, and list the databases for which we collect labeled insight pairs as training data.

Sampling insights. We sample *multiple* insights from each cluster instead of just one to increase the number of insights that users directly label. This does not affect our data collection method except for the final step. That is, when constructing S_2 in Eq (2), instead of considering only one insight from each cluster, we perform a "majority vote": we label each I_i to be more interesting than each I_j only if a majority of insights sampled from C_i are annotated to be more interesting by users than samples from C_j.

Annotation process. We use Amazon Mechanical Turk (MTurk) as the crowdsourcing platform for human worker annotation. Each annotation task contains one question: "Which of the following two insights is more interesting to you?" We also provide the following information in each annotation task:

- *Description*: we provide a short description of the database.
- *Database schema*: we show all the columns with their column names, data types, as well as descriptions of the columns.
- *A pair of insights*: we provide the following information for each insight I: (a) I itself (i.e., a table), (b) the query P producing I, (c) a natural language description of P, and (d) a visualization of I.

We ask MTurk workers to rank insights based on *only* the insight rather than the corresponding query, and use visualizations to help workers gain an intuitive understanding of the insights.

Databases. We labeled training data with 10 real-world databases. More statistics about these databases can be found in Table 1.

5.2 MCMC-based Insight Search

In this section, we present some important implementation details of our insight search algorithm, including a few key optimizations to speed up the search process.

Prefetching query results. Metropolis-Hastings algorithm calculates the acceptance probability α in every iteration , which requires first executing a proposal query to obtain its resulting insight I and then invoking our interestingness measure model on I. However, executing all these queries imposes a significant runtime overhead because our algorithm sometimes needs to run

Table 1: Statistics of databases. Here, the "# queries" column essentially gives the size of the search space.

Database	# columns	# tuples	# queries
Population	5	5,280	23,583
Austin	18	116,675	197,563
Names	5	846,459	191,176
PPI	10	61,968	160,745
Video	10	16,599	31,840
Netflix	6	585	7,581
Absenteeism	13	8,3360	36,852
Marketing	28	2,240	89,190
Flight	12	5,819,080	67,145
Happiness	12	158	4,671

```
SELECT    census_tract, status, COUNT(*)
FROM      Austin Crime
GROUP BY  census_tract, status
ORDER BY  COUNT(*)
```

Figure 8: Query obtained after applying "*grouping where*" to the query from Figure 7a.

hundreds of iterations. Our observation is that many proposals only differ slightly. For instance, consider the proposal query shown in Figure 7b proposed from the query in Figure 7a. Based on this observation, we implemented an optimization that prefetches results of queries that can be inferred from a single query. Our general idea is to generate a query P', given the current query P, such that, using the execution result of P', we can *more cheaply* generate results of *multiple* queries that may be proposed in the future. For instance, one strategy we used is called *"grouping where"*. As shown in Figure 8, we generate P' by first getting rid of the **WHERE** clause from P, and then adding the column used in **WHERE** (in this example, *census_tract*) to **SELECT** and **GROUP BY** clauses. Everything else remains the same. Intuitively, this new query P' computes a table I' with an additional column *census_tract* on which P' performs **GROUP BY**. Then, we split I' into multiple tables, each of which has one unique *census_trace* value. These tables are cached. Therefore, if in the future there is a proposal (such as the one in Figure 7b) that only differs from P in the value used in **WHERE**, we can efficiently obtain its result from cache instead of executing the proposal.

Batching interestingness queries. On top of the previous optimization, another optimization we implement is to eagerly obtain the interestingness scores of all tables resulting from the aforementioned I', instead of querying the machine learning model when there is a proposal that hits our cache in the future. The reason is that, for ML models, batching queries can significantly reduce the inference time, especially on modern GPUs. Therefore, once we split I' into multiple tables, we immediately invoke our neural interestingness model on these tables.

Parallelizing insight search. A final and natural optimization is, parallelizing the MCMC algorithm by running multiple instances. We keep the insight with the highest score across multiple threads.

6 EVALUATION

In this section, we describe a series of experiments that are designed to answer the following research questions:

- **Q1:** Does DAISY actually produce more diverse insights than previous work?
- **Q2:** How well does DAISY's *neural interestingness measure* perform in practice? For example, can it accurately distinguish genuinely interesting insights from less interesting ones?
- **Q3:** How well does DAISY's *search algorithm* work in practice? For instance, can it find insights that are highly ranked by a given interestingness measure within a short amount of time?
- **Q4:** How well does DAISY perform *end-to-end* in practice? For example, can it generate interesting insights that prior techniques are not able to identify?

6.1 Q1: Diversity

In the introduction, we raise the problem that previous systems can only produce interesting insights w.r.t. the system designer's perspective, and fail to generate other insights that would be interesting to a broader audience (e.g. the insights shown in Figure 3). In this subsection, we will show that DAISY can produce a more diverse set of interesting insights to the user.

Setup. We first generate top-k insights with DAISY, then classify each insight into two categories: *possible* or *impossible* to be found by QuickInsights. The classification is performed by leveraging the pruning rules defined in QuickInsights [12]. If an insight does not meet the rules, we classify it as "*impossible* to be found by QuickInsights"; otherwise, it is classified as *possible*. Specifically, we utilize the following rules:

- The leading value in one column dominates *more than 50%* of values in the same column.
- The impact[5] should be larger than a given threshold.

Main result. The main result is presented in Table 2.

Discussion. The result clearly shows that most of the insights generated by DAISY are *impossible* to be found by QuickInsights. Only 7% to 27% of top-ranking insights generated by DAISY are even within the vocabulary of insights that could be generated by QuickInsights. In other words, DAISY is able to generate a more diverse set of insights and cover a broader spectrum of interestingness.

Table 2: The percentage of k insights generated by DAISY that is *impossible* to be found by QuickInsights.

Databases	k =	10	100	1000
Netflix		0.9	0.86	0.93
Video		0.8	0.73	0.75

6.2 Q2: Evaluating Interestingness Measure

We follow the standard k-fold cross-validation method to evaluate our neural interestingness measure.

Setup. We first *randomly* shuffle the ten databases (for which we obtained labeled insights; see Section 5.1), which are then split

[5]the impact score from QuickInsights is calculated based on the number of tuples in the original database used to generate the insight I.

Table 3: Test accuracy (in percentage) for (1) full-fledged neural interestingness measure, (2) variants using different training features (from two sets of features), and (3) variants using different types of training data (namely, S_1, S_2, S_3). We consider five training sets each of which consists of two (randomly selected) databases.

Databases for Testing	Full	Varying training features		Varying data types	
		(a)	(b)	S_1	$S_1 + S_2$
Population + Austin	**71.8**	68.4	71.7	62.9	70.6
Name + PPI	**79.6**	71.4	78.2	65.2	78.5
Video + Netflix	**80.7**	70.5	79.4	68.7	79.9
Absenteeism + Marking	80.3	70.9	**81.5**	64.9	77.3
Flight + Happiness	**78.4**	72.3	77.5	66.3	78.2
Average	**78.2**	70.7	77.7	65.6	76.9

into five disjoint groups where each group has two databases. For each group, we take the human-labeled insights corresponding to databases in that group as our *test set*, and we use the insight pairs corresponding to the remaining eight databases as our *training set*.[6] In other words, we perform a 5-fold cross-validation where our split is at the database level. Given a training set, we train our interestingness measure M using features described in Section 3.2. During testing, given an insight pair (I, I') where I is labeled more interesting than I', if $M(I) > M(I')$, we say M predicts accurately on (I, I'). We report the accuracy on *all* insight pairs in the test set.

Main results. Our main results are given in the "Full" column of Table 3. Overall, our neural interestingness measure achieved an average of 78.2% accuracy across five test sets. Note that databases that we test on are totally different from those in the corresponding training set. This highlights that it is indeed feasible to train interesting measures that can generalize well to unseen databases.

Discussion. In order to better understand the scenarios in which our model would "fail", we manually inspected some insight pairs for which our model did not predict accurately. We identified two common reasons that lead to inaccurate predictions. First, there are some insight pairs in the test set for which we do not have sufficient training data. As a result, the model did not learn how to score those insights well. Second, the training data may contain noise. Two pairs of "similar" insights from two databases may have opposite labels in the training set. As a result, this would confuse the model and lead to wrong predictions.

Ablation studies. We also perform ablation studies to evaluate the relative importance of various design choices. In particular, we consider the following variants in our ablation studies:

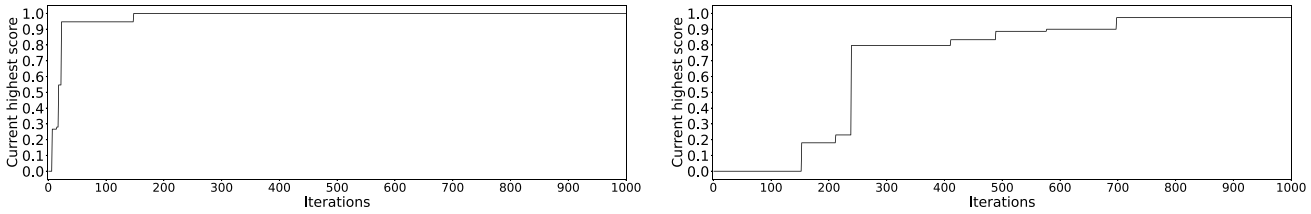
- *Varying training Features*: We split all features described in Section 3.2 into two disjoint groups: (a) *tuple-count + value-sum + max-min + s-dev*, (b) *top-10*. Then, we construct 2 variants that use only the corresponding group of features, perform the same 5-fold cross-validation test (as in previous experiment) and record the accuracy.
- *Varying training data types*: We also consider variants that use *different kinds* of training data. Recall that Section 2 described 3 types of data: S_1 containing insight pairs that are *directly* labeled by human annotators, S_2 consisting of insights pairs whose labels are *inferred* based on our clustering result, and S_3 including *similar insights* from the same cluster. To evaluate the usefulness of different types of training data, we construct 2 variants (one using S_1; the other using S_1 and S_2). For each variant, we conduct the same 5-fold cross-validation test and report the accuracy.
- *Training data amount*: Finally, we consider variants using *different amounts* of training data. In particular, we construct variants that are trained from k databases, where k ranges from 9 to 6. For each k, we *randomly* select k databases for training, and we use the remaining databases for testing. We repeat this process three times. Note that we do not consider *all* combinations of train/test split for a given k because there are too many of them. On the other hand, we believe sampling three combinations should give us a sufficiently good understanding of how different variants work. Finally, we obtained 12 variants.

Table 4: Test accuracy using different training sets.

k	Databases for Testing	Accuracy
$k = 9$	*Austin*	81.3
	Netflix	81.0
	Video	80.4
$k = 8$	*Population + Netflix*	81.3
	Name + Netflix	79.2
	Austin + Marking	80.3
$k = 7$	*Austin + Video + Name*	77.9
	Name + PPI + Flight	76.1
	Marking + Flight + Netflix	74.3
$k = 6$	*Flight + Names + Austin + Marketing*	68.7
	Austin + Flight + PPI + Happiness	67.4
	Names + Absenteeism + Marketing + PPI	66.9

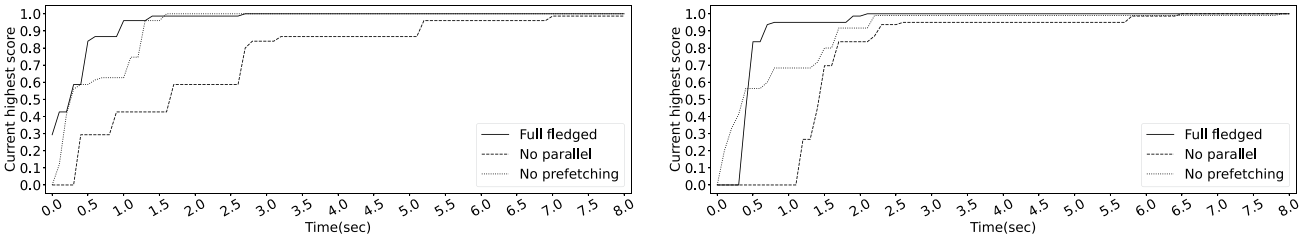
Ablation study results. We summarize the main results in Table 3 ("Varying training features" and "Varying data types" columns) and in Table 4. Now let us take a closer look at these results. First of all, comparing the results in the "Full" column and those in the "Varying data types" column, our main takeaway is that it is critical to use all three types of training data (namely, directly labeled data S_1, inferred data S_2, and similar insight pairs S_3) in order to achieve the highest accuracy. Furthermore, comparing the results from the

[6]Since we aim to test our model's performance on data that is *directly* provided by humans, our test set only contains insight pairs in S_1 (see Eq. 1) that are *directly* labeled by MTurk workers and does not include insights from S_2 and S_3 that are *inferred* based on our clustering result. On the other hand, our train set includes all labeled data in S_1, S_2, S_3; this is to boost the amount of training data in order to train better models.



(a) *Video.* (b) *Netflix.*

Figure 9: The highest score of the currently discovered insight over iterations, where the y-axis is the score and the x-axis is the number of iterations. We run our search algorithm 3 times and plot the mean.



(a) *Video.* (b) *Netflix.*

Figure 10: Results for ablation studies. Note that now x-axis gives the running time.

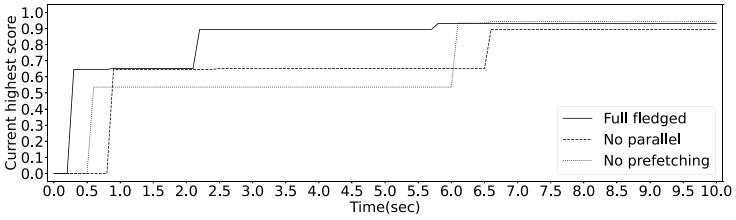


Figure 11: Search performance on *Flight* dataset.

"Full" column with those from the "Varying training features" column, we conclude that it is also beneficial to include all training features for the model to generalize better. Finally, looking at Table 4, our observation is that we can train a fairly accurate neural interestingness measure with eight databases. With less training data, the model performance would get noticeably worse. On the other hand, if we increase the amount of training data, the model performance would increase though only slightly.

6.3 Q3: Evaluating Search Algorithm

The central question we aim to answer when evaluating our MCMC-based search technique is: can our search algorithm discover highly ranked insights according to a given interestingness measure?

Setup. Since our search algorithm is agnostic to the underlying interestingness measure, we conducted this experiment using a randomly selected neural interestingness measure M trained from eight databases; we picked the third one from Table 3. We then evaluate our MCMC-based search algorithm using this M on the remaining two databases, i.e., *Video* and *Netflix*. In particular, for each database, we run our search algorithm for 1,000 iterations and record the interestingness score of the current highest-ranked insight at the end of each iteration. Since our search algorithm is stochastic, we run it three times and record the average score in each iteration. Furthermore, to understand how close our generated insight is to the *globally best insight* in the entire search space, we also obtained the globally highest interestingness score by running our interestingness measure on *every* insight in our query language. We normalize scores of the discovered insights to this globally highest score.

Main results. Figure 9 summarizes our main results. As we can see, overall, our search algorithm is able to discover insights with very high scores within a few hundred iterations. For instance, for the *Video* database, `DAISY` found an insight with a (normalized) score 0.95 after 25 MCMC iterations. For the *Netflix* database, it generated an insight with a score of 0.9 after 580 iterations. While this result may look noticeably "worse" than the result for *Video*, it is still pretty fast regarding the actual running time. In particular, for both *Video* and *Netflix*, it took at most 2 seconds for the algorithm to converge to an insight with a score higher than 0.9.

Discussion. Note that the search space for *Video* contains significantly more queries than that of *Netflix* (16,599 vs. 585; see the "**# queries**" column in Table 1). However, our search algorithm found high-score insights for *Video* using significantly fewer iterations than for *Netflix* (25 vs. 580). This seemingly contradictory result is due to the higher density of interesting insights in *Video*. While having a much larger search space, *Video* contains significantly more queries that yield interesting insights. Therefore, overall, it is easier for our technique to discover interesting insights for *Video* than for *Netflix* within fewer iterations.

Ablation studies. We consider the following variants to evaluate the relative importance of our optimizations:

- *No prefetching*: this is a variant of our search algorithm without the "prefetching query results" optimization.
- *No parallel search*: this is a variant that does not run multiple MCMC search instances in parallel; it has only one thread.

For each variant, we conduct the same experiment described above. However, instead of recording the scores at the end of each iteration, we record scores *every 0.1 seconds* to compare the running times of all three techniques.

Ablation study results. Figure 10 summarizes our main results. Our main take-away message is that both optimizations are important to the final performance of our search algorithm. More specifically, for *Video*, the full-fledged technique was able to find an insight with a score higher than 0.9 within 1 second. However, using the variants typically would take a few more seconds. We observed very similar patterns for *Netflix* as well.

Discussion. While *Netflix* required significantly more iterations than *Video* to discover an interesting insight (580 vs. 25, see Figure 9), their running times are pretty close (both around 1 second, see Figure 10). This is because queries from *Video* generally take longer time to execute, which "slows down" its search process.

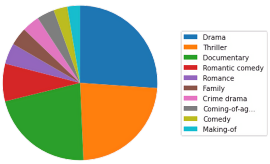
6.3.1 Search Algorithm Scalability. To evaluate the scalability of the search algorithm, we also report the performance and runtime space overhead on the largest dataset, *Flight*.

Setup. The interestingness measure model we use for this experiment is the fifth one listed in Table 3. We monitor the memory usage during the search phase. We record the time for generating the insights with QuickInsights on the *Flight* dataset.

Result. As shown in Figure 11, DAISY takes around 5.5 seconds to find an insight with a score higher than 0.9 with the full fledged model. QuickInsights took 20 seconds to generate the top-40 insights on a sampled subset of *Flight* dataset.[7] The maximum memory usage of the search phase is around 300MiB.

Discussion. Compared with the two smaller datasets *Video* and *Netflix*, DAISY takes a longer time to find top ranked insights. However, we argue that it is still a reasonable time for a large dataset like *Flight*, given that QuickInsights took 20 seconds to generate the insights on a sampled subset of the *Flight* dataset. Besides, the memory consumption of the search phase is also reasonable given the size of the dataset (5,819,080 tuples; 565MB of raw data) and the search space (67,145 of queries). This experiment illustrates that DAISY scales to larger datasets, exhibiting efficiency in terms of both time and space.

[7] QuickInsights on Microsoft PowerBI automatically samples a subset for large datasets.




```
SELECT    Genre, SUM(Runtime)
FROM      Netflix
WHERE     Language="Spanish"
GROUP BY  Genre
ORDER BY  SUM(Runtime) DESC
```

Figure 12: An interesting insight generated by DAISY.

Table 5: Summary of user study results.

Database	System	Mean	Max	Min	p-value
Netflix	DAISY	7.1	9	6	0.005
	QuickInsights	5.7	7	4	
Video	DAISY	7.6	9	6	0.021
	QuickInsights	6.6	8	5	

Table 6: Interestingness score distribution.

Score range	[0, 0.2)	[0.2, 0.4)	[0.4, 0.6)	[0.6, 0.8)	[0.8, 0.9)	[0.9, 1]
% (Netflix)	0.892	0.089	0.007	0.009	0.001	0.002
% (Video)	0.599	0.069	0.114	0.115	0.059	0.044

6.4 Q4: Evaluating DAISY End-to-end

One of the main goals of our work is to be able to generate *broadly* interesting insights, the key question we aim to address, when evaluating DAISY end-to-end, is whether or not it can discover insights that are more broadly interesting than prior techniques, such as QuickInsights. To answer this question, we conducted a user study for both DAISY and QuickInsights.

User study setup. We recruited 20 participants through Amazon Mechanical Turk who works in the "Software & IT Services" industry. Each participant took a survey prepared by us. In particular, the survey first shows two sets of insights: (1) a set of 10 (visualized) insights discovered by DAISY for a database, and (2) another set of 10 insights identified by QuickInsights for the same database.

Here, for QuickInsights, we used the "get Quick Insights" feature in Microsoft PowerPI and retained its top 10 insights. For DAISY, we ran our full-fledged technique on the database for 30 seconds and retained the top 10 insights. More specifically, DAISY used the interestingness measure trained from all databases except for *Video* and *Netflix*. In other words, in this user study, we consider two "test databases" (namely, *Video* and *Netflix*). Therefore, in our user study, each participant would see a total of 20 insights for either *Video* or *Netflix*, 10 generated by DAISY, and 10 generated by QuickInsights.

The survey asks the following question for each of the two sets of insights: "How interesting are these insights to you?" We ask the participant to provide an overall score on a scale of 1-10, based on the ten insights they can see for a given database. We record all the ratings from each participant for this question.

Finally, to familiarize the participant with the database, our survey also includes some information, such as a short description and the schema of the database as well as record samples.

User study results. Our main results are summarized in Table 5. The high-level message is that DAISY is able to generate insights

that users indeed found interesting. Furthermore, we ran a standard 1-tailed t-test to evaluate whether our user study results were statistically significant by calculating the p-value (shown in Table 5). We can see that the p-values are less than 0.05 for both test databases, meaning our results are statistically significant. Therefore, we believe our proposed technique can discover more interesting insights than QuickInsights. Finally, we manually inspected the insights generated by DAISY and found that they cover a broader class of types than those identified by QuickInsights. For example, Figure 12 shows an insight discovered by DAISY with two dominating values (Drama and Thriller). While neither of them dominates more than 50%, participants in our user study still found it very interesting. We believe this demonstrates the advantage of our data-driven approach over prior rule-based insight generation techniques.

Comparison with QuickInsights. In addition to a user study, we also compared DAISY with QuickInsights using an experiment that is designed to answer the following question: can DAISY "cover" insights generated by QuickInsights? In particular, in this experiment, we first ran QuickInsights on *Netflix* and obtained its top 10 insights. Then, we treat these 10 insights as the "gold" insights and test whether DAISY can find them. Our main result is that, with 2 seconds, DAISY is able to find 8 (out of 10) insights. Furthermore, the average (normalized) score DAISY assigned to these 8 insights is 0.94. We believe this result highlights that DAISY can pretty well cover those insights from QuickInsights.

Discussion. As we can see, DAISY was able to discover insights with very high interestingness scores. It also covered most of the insights that QuickInsights can generate. One might wonder: is that because DAISY's interestingness measure always assigns a high score for every insight in the search space? To answer this question, we performed a small experiment and report the interestingness score distribution in Table 6. As we can see, only 0.2% of insights in the entire space have scores higher than 0.9 on *Netflix*, whereas almost 90% have shallow interestingness scores (lower than 0.2). We observed similar patterns for *Video* overall as well, though *Video* has a higher density of interesting insights.

7 RELATED WORK

In this section, we discuss closely related work on insight discovery, visualization recommendation, and program synthesis.

Insight discovery for multi-dimensional data. There has been a long line of work on mining different types of insights and modeling different perspectives on "interestingness" from multi-dimensional data [11, 19, 21, 25, 30, 32, 39, 44, 50, 54, 55]. For example, Knorr and Ng [27] focused on mining outliers from large datasets. Sarawagi [37] proposed to explore "surprising" parts of OLAP data, and the "surprisingness" is calculated based on the Maximum Entropy principle. To the best of our knowledge, QuickInsights [12] is the most recent work that provides a unified formulation of interesting insights of 12 different types and an efficient mining algorithm that leverages multiple optimization and pruning techniques based on their assumptions.

In this work, instead of using hand-crafted interestingness measures, we proposed to take a *data-driven* approach that aim to *learn* how interesting a data pattern is from the data. Thus, many optimization techniques developed in previous work cannot be applied to our work. Instead, we formulate the insight discovery problem as an optimal program synthesis problem and propose a solution based on stochastic search to generate interesting insights. We note that most of the prior techniques require knowing the internal structures of the interestingness measure. In contrast, our approach is agnostic to the underlying structure of the interestingness measure.

Interestingness measure for insight discovery. As mentioned earlier, a key distinction of our work from prior techniques is that our interestingness measure is automatically learned from data instead of being designed by developers. For instance, the interestingness measure in QuickInsights [12] consists of two key metrics: impact and significance. Here, impact roughly corresponds to the fraction of the input database that an insight is generated from: the larger this fraction is, the more interesting the insight is. Significance is a statistical measure that reflects how significant the insight is against a defined null hypothesis. MetaInsight [31] includes additionally homogeneous data patterns to enrich the pattern types. Other prior works have also proposed different metrics to measure the interestingness of data patterns. For example, Geng and Hamilton [16] identified nine criteria: coverage, surprisingness, diversity, actionability, etc. In contrast to these techniques, most of which use *rule-based* interestingness measures, DAISY's interestingness measure is a neural network trained from data.

Program synthesis for databases. Program synthesis has been applied in database research. One important task is generating SQL queries automatically from high-level specifications such as examples and natural language [3, 4, 23, 26, 29, 36, 38, 42, 53]. For instance, SQLizer [52] uses an enumerative search algorithm combined with semantic parsing that can synthesize SQL queries from English sentences. Similarly, Scythe [46] generates SQL queries from high-level specifications which, different from SQLizer, are based on input-output examples. In addition to SQL synthesis, prior work has also explored other applications within the domain of databases, such as data migration [49, 51], data wrangling [24, 40, 41, 47, 48], data discovery [35], and data summarization [13].

Optimal program synthesis. Our work formulates the insight generation problem as an *optimal* program synthesis problem. This is a new application that has not yet been explored by prior work on optimal program synthesis [7, 34].

8 CONCLUSION AND FUTURE WORK

In this paper, we presented a system called DAISY that can automatically identify interesting insights for multi-dimensional data. Different from prior work, DAISY uses a data-driven insight synthesis approach by training a neural network from data that can quantitatively score the interestingness of a given insight. Then, given this neural interestingness measure, DAISY uses an MCMC-based stochastic search algorithm to generate interesting insights effectively. Our technique employs a new problem formulation that is based on optimal program synthesis. Our evaluation results show that DAISY can effectively synthesize broadly interesting insights and significantly advances the state-of-the-art.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 1934565, 2106176 and 2312931.

REFERENCES

[1] Austin Crime Dataset. https://www.kaggle.com/jboysen/austin-crime.

[2] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. *Syntax-guided synthesis.* IEEE.

[3] Christopher Baik, H. V. Jagadish, and Yunyao Li. 2019. Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases. In *ICDE*. IEEE, 374–385.

[4] Christopher Baik, Zhongjun Jin, Michael J. Cafarella, and H. V. Jagadish. 2020. Duoquest: A Dual-Specification System for Expressive SQL Queries. In *SIGMOD Conference*. ACM, 2319–2329.

[5] J Belissent, E Cullen, G Leganza, and J Lee. 2019. Gartner identifies top 10 data and analytics technology trends for 2019.

[6] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2021. Deep Neural Networks and Tabular Data: A Survey. *CoRR* abs/2110.01889 (2021).

[7] James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. 2016. Optimizing synthesis with metasketches. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 775–788.

[8] David CeArley, Brian Burke, Samantha Searle, and Mike Walker. 2018. Top 10 strategic technology trends for 2018.

[9] Thomas Cleff. 2014. Exploratory data analysis in business and economics. *Exploratory Data Analysis in Business and Economics. https://doi. org/10.1007/978-3-319-01517-0* (2014).

[10] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[11] Çagatay Demiralp, Peter J. Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. 2017. Foresight: Recommending Visual Insights. *Proc. VLDB Endow.* 10, 12 (2017), 1937–1940.

[12] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. QuickInsights: Quick and Automatic Discovery of Insights from Multi-Dimensional Data. In *SIGMOD Conference*. ACM, 317–332.

[13] Anna Fariha, Matteo Brucato, Peter J Haas, and Alexandra Meliou. 2020. SuDocu: summarizing documents by example. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2861–2864.

[14] Yanwei Fu, Timothy M. Hospedales, Tao Xiang, Jiechao Xiong, Shaogang Gong, Yizhou Wang, and Yuan Yao. 2016. Robust Subjective Visual Property Prediction from Crowdsourced Pairwise Labels. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 3 (2016), 563–577.

[15] Johannes Fürnkranz and Eyke Hüllermeier. 2010. Preference Learning and Ranking by Pairwise Comparison. In *Preference Learning*. Springer, 65–82.

[16] Liqiang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Comput. Surv.* 38, 3 (2006), 9.

[17] Lukas Gienapp, Benno Stein, Matthias Hagen, and Martin Potthast. 2020. Efficient Pairwise Annotation of Argument Quality. In *ACL*. Association for Computational Linguistics, 5772–5781.

[18] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. 1995. *Markov chain Monte Carlo in practice.* CRC press.

[19] Jiawei Han and Micheline Kamber. 2000. *Data Mining: Concepts and Techniques.* Morgan Kaufmann.

[20] W. K. Hastings. 1970. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika* 57 (1970), 97–109.

[21] Kevin Zeng Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *CHI*. ACM, 128.

[22] Qinheping Hu and Loris D'Antoni. 2018. Syntax-guided synthesis with quantitative syntactic objectives. In *International Conference on Computer Aided Verification*. Springer, 386–403.

[23] Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural Language to Structured Query Generation via Meta-Learning. In *NAACL-HLT (2)*. Association for Computational Linguistics, 732–738.

[24] Zhongjun Jin, Michael R. Anderson, Michael J. Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *SIGMOD Conference*. ACM, 683–698.

[25] Daniel A. Keim. 2002. Information Visualization and Visual Data Mining. *IEEE Trans. Vis. Comput. Graph.* 8, 1 (2002), 1–8.

[26] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proc. VLDB Endow.* 13, 10 (2020), 1737–1750.

[27] Edwin M Knorr and Raymond T Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, Vol. 98. Citeseer, 392–403.

[28] Matthieu Komorowski, Dominic C. Marshall, Justin D. Salciccioli, and Yves Crutain. 2016. *Exploratory Data Analysis.* Springer International Publishing, Cham, 185–203. https://doi.org/10.1007/978-3-319-43742-2_15

[29] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query From Examples: An Iterative, Data-Driven Approach to Query Construction. *Proc. VLDB Endow.* 8, 13 (2015), 2158–2169.

[30] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *ICDE*. IEEE Computer Society, 101–112.

[31] Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. 2021. MetaInsight: Automatic Discovery of Structured Knowledge for Exploratory Data Analysis. In *SIGMOD Conference*. ACM, 1262–1274.

[32] Andrew M. McNutt, Anamaria Crisan, and Michael Correll. 2020. Divining Insights: Visual Analytics Through Cartomancy. In *CHI Extended Abstracts*. ACM, 1–16.

[33] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics* 21, 6 (1953), 1087–1092.

[34] Shankara Pailoor, Xinyu Wang, Hovav Shacham, and Isil Dillig. 2020. Automated policy synthesis for system call sandboxing. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–26.

[35] El Kindi Rezig, Anshul Bhandari, Anna Fariha, Benjamin Price, Allan Vanterpool, Vijay Gadepally, and Michael Stonebraker. 2021. DICE: data discovery by example. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2819–2822.

[36] El Kindi Rezig, Allan Vanterpool, Vijay Gadepally, Benjamin Price, Michael J. Cafarella, and Michael Stonebraker. 2020. Towards Data Discovery by Example. In *Poly/DMAH@VLDB (Lecture Notes in Computer Science)*, Vol. 12633. Springer, 66–71.

[37] Sunita Sarawagi. 2000. User-adaptive exploration of multidimensional data. In *VLDB*, Vol. 2000. Citeseer, 307–316.

[38] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. 2014. Discovering queries based on example tuples. In *SIGMOD Conference*. ACM, 493–504.

[39] Danqing Shi, Xinyue Xu, Fuling Sun, Yang Shi, and Nan Cao. 2021. Calliope: Automatic Visual Data Story Generation from a Spreadsheet. *IEEE Trans. Vis. Comput. Graph.* 27, 2 (2021), 453–463.

[40] Rishabh Singh. 2016. BlinkFill: Semi-supervised Programming By Example for Syntactic String Transformations. *Proc. VLDB Endow.* 9, 10 (2016), 816–827.

[41] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *Proc. VLDB Endow.* 11, 2 (2017), 189–202.

[42] Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic Parsing with Syntax- and Table-Aware SQL Generation. In *ACL (1)*. Association for Computational Linguistics, 361–372.

[43] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *SIGMOD Conference*. ACM, 1509–1524.

[44] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. 2015. SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *Proc. VLDB Endow.* 8, 13 (2015), 2182–2193.

[45] Nico Verbeeck, Richard M Caprioli, and Raf Van de Plas. 2020. Unsupervised machine learning for exploratory data analysis in imaging mass spectrometry. *Mass spectrometry reviews* 39, 3 (2020), 245–291.

[46] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive SQL queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 452–466.

[47] Xinyu Wang, Sumit Gulwani, and Rishabh Singh. 2016. FIDEX: filtering spreadsheet data using examples. In *OOPSLA*. ACM, 195–213.

[48] Yuepeng Wang, James Dong, Rushi Shah, and Isil Dillig. 2019. Synthesizing database programs for schema refactoring. In *PLDI*. ACM, 286–300.

[49] Yuepeng Wang, Rushi Shah, Abby Criswell, Rong Pan, and Isil Dillig. 2020. Data Migration using Datalog Program Synthesis. *Proc. VLDB Endow.* 13, 7 (2020), 1006–1019.

[50] Yun Wang, Zhida Sun, Haidong Zhang, Weiwei Cui, Ke Xu, Xiaojuan Ma, and Dongmei Zhang. 2020. DataShot: Automatic Generation of Fact Sheets from Tabular Data. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020), 895–905.

[51] Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. 2018. Automated migration of hierarchical data to relational tables using programming-by-example. *Proceedings of the VLDB Endowment* 11, 5 (2018), 580–593.

[52] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–26.

[53] Sai Zhang and Yuyin Sun. 2013. Automatically synthesizing SQL queries from input-output examples. In *ASE*. IEEE, 224–234.

[54] Mengyu Zhou, Qingtao Li, Xinyi He, Yuejiang Li, Yibo Liu, Wei Ji, Shi Han, Yining Chen, Daxin Jiang, and Dongmei Zhang. 2021. Table2Charts: Recommending Charts by Learning Shared Table Representations. In *KDD*. ACM, 2389–2399.

[55] Mengyu Zhou, Wang Tao, Pengxin Ji, Han Shi, and Dongmei Zhang. 2020. Table2Analysis: Modeling and Recommendation of Common Analysis Patterns for Multi-Dimensional Data. In *AAAI*. AAAI Press, 320–328.