Automatic generation of comparison notebooks for interactive data exploration

Alexandre Chanson
University of Tours
Blois, France
Alexandre.Chanson@univ-tours.fr

Nicolas Labroche
University of Tours
Blois, France
Nicolas.Labroche@univ-tours.fr

Patrick Marcel
University of Tours
Blois, France
Patrick.Marcel@univ-tours.fr

Stefano Rizzi
University of Bologna
Bologna, Italy
Stefano.Rizzi@unibo.it

Vincent T'Kindt
University of Tours
Tours, France
Vincent.Tkindt@univ-tours.fr

ABSTRACT

We consider the problem of generating SQL notebooks of comparison queries for Exploratory Data Analysis (EDA). A comparison query allows to find insights in a dataset by specifying the comparison of subsets of data. In this paper, we study the problem of generating sequences of comparison queries that are insightful and coherent. We propose exact and approximate resolution approaches, and study their efficiency and effectiveness on artificial and real datasets, as well as with a user study.

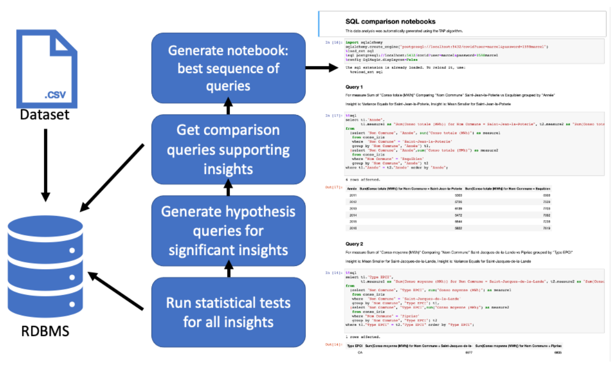
1 INTRODUCTION

Imagine a data enthusiast with some basic knowledge of SQL, having to explore an unknown open data set in CSV format. Generating meaningful insights from this dataset is tedious, because it requires sending many queries for data profiling, formulating hypothesis, computing comparisons, etc. — not to mention running many statistical tests to ensure insights' significance.

In this work, we contribute to the field of automatic generation of data exploration sessions (see e.g., [11, 25]). Specifically, we study the problem of generating meaningful sequences of comparison insights over a potentially unknown dataset, which could serve as an entry point in the exploration of this dataset.

We work with the following hypotheses. Firstly, we assume that the dataset consists of one table, imported into a RDBMS, for which the user only has to distinguish between numeric attributes and categorical attributes before starting to query it with SQL. While the term EDA session is frequently associated with languages like Python, we insist on the importance of SQL for data workers, as illustrated by the 10,000+ data professionals who responded to StackOverflow's 2020 survey[1] showing that SQL is the most used language in data science.

Secondly, we assume the user is interested in comparisons. Comparison are extremely popular among data workers [4, 24, 29]. In this work, we define comparison queries as queries used to compare two data series. We also assume that the user is interested in a sequence of such comparison queries, which we will call a *comparison notebook* in what follows.


.csv
Dataset
RDBMS
Generate notebook: best sequence of queries
Get comparison queries supporting insights
Generate hypothesis queries for significant insights
Run statistical tests for all insights

SQL comparison notebooks

This data analysis was automatically generated using the TAP algorithm.

```
In [16]: import sqlalchemy
sqlalchemy.create_engine("postgresql://localhost:5432/covid?user=marcel&password=1598marcel")
%load_ext sql
%sql postgresql://localhost:5432/covid?user=marcel&password=1598marcel
%config SqlMagic.displaycon=False

The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

Query 1

For measure Sum of "Conso totale (MWh)" Comparing "Nom Commune" Saint-Jean-la-Poterie vs Esquibien grouped by "Année"

Insight is: Variance Equals for Saint-Jean-la-Poterie, Insight is: Mean Smaller for Saint-Jean-la-Poterie

```
In [17]: %%sql
select t1."Année",
       t1.measure1 as "Sum(Conso totale (MWh)) for Nom Commune = Saint-Jean-la-Poterie", t2.measure2 as "Sum(Conso tot
from
  (select "Nom Commune", "Année", sum("Conso totale (MWh)") as measure1
   from conso_iris
   where  "Nom Commune" = 'Saint-Jean-la-Poterie'
   group by "Nom Commune", "Année") t1,
  (select "Nom Commune", "Année",sum("Conso totale (MWh)") as measure2
   from conso_iris
   where "Nom Commune" = 'Esquibien'
   group by "Nom Commune", "Année") t2
where t1."Année" = t2."Année" order by "Année";

6 rows affected.
```

Out[17]:

Année	Sum(Conso totale (MWh)) for Nom Commune = Saint-Jean-la-Poterie	Sum(Conso totale (MWh)) for Nom Commune = Esquibien
2011	5303	6968
2012	5730	7509
2013	6139	7708
2014	5472	7082
2015	5644	7238
2016	5822	7019

Query 2

For measure Sum of "Conso moyenne (MWh)" Comparing "Nom Commune" Saint-Jacques-de-la-Lande vs Pipriac grouped by "Type EPCI"

Insight is: Mean Smaller for Saint-Jacques-de-la-Lande, Insight is: Variance Equals for Saint-Jacques-de-la-Lande

```
In [14]: %%sql
select t1."Type EPCI",
       t1.measure1 as "Sum(Conso moyenne (MWh)) for Nom Commune = Saint-Jacques-de-la-Lande", t2.measure2 as "Sum(Conso
from
  (select "Nom Commune", "Type EPCI", sum("Conso moyenne (MWh)") as measure1
   from conso_iris
   where  "Nom Commune" = 'Saint-Jacques-de-la-Lande'
   group by "Nom Commune", "Type EPCI") t1,
  (select "Nom Commune", "Type EPCI",sum("Conso moyenne (MWh)") as measure2
   from conso_iris
   where "Nom Commune" = 'Pipriac'
   group by "Nom Commune", "Type EPCI") t2
where t1."Type EPCI" = t2."Type EPCI" order by "Type EPCI";

1 rows affected.
```

Out[14]:

Type EPCI	Sum(Conso moyenne (MWh)) for Nom Commune = Saint-Jacques-de-la-	Sum(Conso moyenne (MWh)) for Nom Commune = Pipriac
CA	6077	6805

Figure 1: Overview of the approach

Finally, we assume that the user expects the comparison insights drawn from the dataset to be statistically significant. Consistently with prior approaches [25, 29], we consider that insights should be validated using appropriate statistical tests.

An overview of our approach is presented in Figure 1. Once the dataset is loaded in the RDBMS, a series of statistical tests are performed to select the significant insights. These are then turned into *hypothesis queries*, i.e., queries that the user would have to write to check whether an aggregate query over the dataset is an evidence of an insight. Then only those aggregate queries that are evidences of insights, that we call *comparison queries*, are retained. Finally, a notebook of comparison queries is generated, by picking a given number of comparison queries that maximize an interestingness criterion and are arranged in a sequence that minimizes a distance between them.

Our contributions include:

- a logical framework for the definition of comparison insights, hypothesis queries, comparison queries, and comparison query interestingness;
- the formal definition of the problem of generating notebooks of comparison queries;
- approaches for computing exact and approximate solutions to this problem;
- implementations of the approaches to generate SQL notebooks of comparison queries; and
- a series of tests on artificial and real datasets, including a user study, to investigate the scalability of the implementations and the accuracy of the approximate solutions.

The paper is organized as follows. Section 2 discusses the related work. Section 3 introduces comparison queries, hypothesis

[1] https://insights.stackoverflow.com/survey/2020

queries, and insights. Section 4 formalizes the problem while Section 5 presents our resolution schemes. Section 6 presents the tests we have done. Section 7 concludes and draws the research perspectives.

2 RELATED WORK

Exploratory Data Analysis, insights. Exploratory Data Analysis (EDA), the notoriously tedious task of interactively analyzing datasets to gain insights, has attracted a lot of attention lately [14]. Supporting this task can be done, e.g., by generating EDA notebooks using deep learning [11] or by pre-analyzing datasets for computing insights [25]. Quantifying the importance of insights also attracted a lot of attention in both the database and visualization communities [25, 29]. It is commonly admitted that interestingness in EDA is manifold [13, 16]. In [11], Bar El et al. distinguish different kinds of interestingness depending on the type of EDA operation: on the one hand, for group-by operation, a conciseness measures considers compact group-by results that covers many tuples as both informative and easy to understand; on the other hand, for filtering operation, a measure of exceptionality compares the filtered tuples to a reference, larger set. The statistical significance of patterns extracted from datasets is often used or mixed with other interestingness measures [25, 29]. Tang et al. [25] proposed an approach for extracting trend insights from multidimensional data (like yearly increasing of sales, etc.). They compute the top-k insights in terms of a measure combining statistical significance and importance of the values displayed in the insights. Ding et al. proposed QuickInsight [10], a system for discovering a broad spectrum of insights (Change point, Correlation, Outlier, Seasonality, etc.) in multidimensional data, with the same approach as [25] for scoring insights. Metainsight [15] aims at organizing the knowledge conveyed by insights extracted from multidimensional data, in terms of commonness and exception. The insight score is a combination of impact (the importance of the data scope against the entire database) and conciseness (entropy-like formula quantification of the generality of the insight). Rojas et al. [18] study the feasibility of using sampling in the context of big data exploration. They conclude that combining different sampling strategies helps grasping a broad and diverse set of insights, especially if no prior knowledge on the data is available.

EDA is similar to *Discovery-Driven Exploration* (DDE) of data cubes [22], in whose context the pioneering works by Sunita Sarawagi [20, 21, 23] proposed techniques for interactively browsing interesting cells in a data cube. DDE was essentially motivated by explaining unexpected data in the result of a cube query. Unexpectedness was characterized in terms of deviation from the uniform distribution [21] or notable discrepancies in the data to be explained by generalization (rolling-up) [23] or by detailing (drilling-down) [20]. In contrast to DDE, EDA does not assume that the dataset explored has the multidimensional model of a cube, nor does it assume that the exploration is limited to classical OLAP operations. EDA operations include data retrieval, data representation, and data mining tasks [17]. Finally, EDA aims at finding insights in the data, i.e., high-level observations that are significant. In our work, the focus is on comparison insights, i.e., significant observations made by comparing two series of data, a type of discovery that has not been addressed by DDE.

Comparisons. Several studies highlighted the importance of comparisons when analyzing data. For instance, Blount et al. [4] examined 67 stories, including award-winning data stories, from both professional journalists and data science-aware students, and found comparisons (showing multiple visualisations juxtaposed and highlighting the difference between them) to be the most popular pattern among novices and professionals alike. Zgraggen et al. [29] study the problem of obtaining spurious comparison insights when exploring a dataset. They define comparison insights as observations, hypotheses, and generalizations directly extracted from data and that did not require prior knowledge or domain expertise. Insights are categorized into five insight classes: shape, mean, variance, correlation, and ranking. Each class has its own hypothesis-testing scheme for insight validation. The authors designed an experiment where participants explore a synthetic dataset and instructed them to report their reliable insights by writing them down. 60% of user-reported insights were spurious, which underlines the need for systems being able to automatically characterize insights. Francia et al. [12] and, independently, Siddiqui et al. [24] respectively defined the Assess and Compare operators to give a clear semantics and logical foundations of comparisons (and labelling the result of the comparison in [12]) of two series of data. While Compare is expressed in SQL and implemented and optimized within a RDBMS, Assess is expressed in terms of a cube algebra and implemented as a middleware.

Automatic generation of data explorations. While the problem of automatic generation of coherent EDA sessions has already been investigated [11], to the best of our knowledge the only tentative formal definition of this problem was given in a previous work of ours [5]. Dubbed the *Traveling Analyst Problem* (TAP), it describes the computation of a sequence of interesting queries over a dataset, given a time budget on the query execution cost, and such that the distance between the queries is minimized. TAP differs from the classical orienteering problem [27] by adding a knapsack constraint to it. In [6] we remark that this formulation is close to that of chain composite item (CCI) retrieval [19] for travel itinerary recommendation, defined in terms of compatibility (e.g., geographic distance), validity (e.g., the total cost of an itinerary is within budget), and maximality (e.g., the itinerary should be of the highest value in terms of its POIs popularities), the latter being often used as the objective function. Retrieval of CCIs is usually NP-hard, being reduced to TSP or orienteering problems. Compared to TAP, CCI usually allows merging action cost and travel time budget, while for TAP the distance has a semantics in itself and cannot be led back to a time or to a physical distance. Furthermore, differently from the classical orienteering problem, starting and ending points are not specified.

Since comparisons happen frequently in practice, with a high risk of comparison-based insights being spurious, there is a need to automate the production of non-spurious comparison insights. To the best of our knowledge, our approach is the only one combining comparison insight identification with notebook generation. Our work complements previous works in EDA [10, 25] that address other forms of insights. Noticeably, while we restrict here to comparison insights, our approach can be extended to other forms of insights; the characterization of the insights supported is left as future work, as briefly discussed in the conclusion. We specifically target a special case of Compare queries identified in [24], namely many-to-many comparison queries over fixed X and Y attributes. We aim at efficiently and automatically extracting non-spurious comparison insights in the sense of Zgraggen et al. [29], on a potentially unknown dataset, and generate EDA sessions consisting of related insights.

3 COMPARISON QUERIES, HYPOTHESIS QUERIES, AND INSIGHTS

This section presents our logical framework. Section 3.1 defines comparison queries and notebooks, and Section 3.2 defines insights and hypothesis queries. Finally, Section 3.3 introduces insight credibility and a transitivity relation over insights.

3.1 Comparison queries

We consider an instance of relation R of schema $R[A_1, \ldots, A_n, M_1, \ldots, M_m]$. The A_i's are categorical attributes and the M_j's are numerical attributes, called *measures* in what follows. The active domain of attribute A is noted $dom(A)$.

Definition 3.1 (Comparison query). Comparison queries are extended relational queries of the form:

$$\tau_A((\gamma_{A,agg(M)\rightarrow val}(\sigma_{B=val}(R))) \bowtie (\gamma_{A,agg(M)\rightarrow val'}(\sigma_{B=val'}(R))))$$

where A, B are categorical attributes in $\{A_1, \ldots, A_n\}$, M is a measure attribute in $\{M_1, \ldots, M_m\}$, agg is an aggregate function, and $val, val' \in dom(B)$. γ denotes the grouping/aggregation operator and τ the sorting operator.

For simplicity, we work under the following assumptions: (i) a single attribute is used to aggregate each query; (ii) a single measure is computed for each query; (iii) all aggregation operators can be applied to all measures; (iv) there is no functional dependency between categorical attributes.[2] In what follows, a comparison query is described by the 6-tuple $(A, B, val, val', M, agg)$. Finally, we call *comparison notebook*, or notebook for short, a finite sequence of comparison queries.

Note that our definition of comparison queries requires that a tabular presentation is used for presentation of the result, hence the need for the join and the projection in the out-most position. Alternatively, comparison queries could be written without join: $\gamma_{A,B,agg(M)}(\sigma_{B=val \vee B=val'}(R))$. However, this would require a pivot operation [8] to present the result in a suitable tabular way. In [12] we experimented with the two forms and they appeared to be similar in terms of execution cost.

Lemma 3.2 (Number of comparison queries). *For a relation R of schema $R[A_1, \ldots, A_n, M_1, \ldots, M_m]$, where f aggregation functions can be applied over the m measures, the number of possible comparison queries is polynomial in the number n of categorical attributes and the cardinalities of their active domains:*

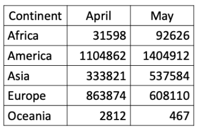
$$\sum_{i=1}^{n} \binom{|dom(A_i)|}{2} \times (n-1) \times m \times f$$

Example 3.3. An example of comparison query for the analysis of COVID-19 infections is given in Figure 2. A comparison notebook could start with the query of that figure, change the measure (replacing cases by deaths), change the aggregation function (replacing sum by avg), then change selection by comparing months 5 and 6, and finally drill continents down to countries (keeping the same selections).

3.2 Insights and hypothesis queries

Consistently with previous characterizations of insights in EDA [25, 29], we see *insights* over a dataset as declarations such as "On average there were more COVID cases in May compared

```
select t1.continent, April, May
from
 (select month, continent, sum(cases) as April
  from covid where month = '4' group by month, continent) t1,
 (select month, continent, sum(cases) as May
  from covid where month = '5' group by month, continent) t2
where t1.continent = t2.continent
order by t1.continent;
```



Continent	April	May
Africa	31598	92626
America	1104862	1404912
Asia	333821	537584
Europe	863874	608110
Oceania	2812	467

Figure 2: A SQL comparison query, its result, and an insight

to April" based on a visual display that triggers the insight, i.e., on the result of a user *comparison query* over the dataset (e.g., number of cases grouped by continents, in April and May). To check the significance of an insight i, i is turned into a testable statistical *hypothesis*, and the significance of i corresponds to the *p-value* of the statistical test. For instance, the insight "On average there were more COVID cases in May compared to April" is turned into the null hypothesis (i.e., assuming the absence of effect) $E[X] = E[Y]$ where X and Y are the random variables representing number of cases for April and May, respectively.

In the case of comparisons, we give a specific definition of insights as declarations concerning two particular values of a given categorical attribute.

Definition 3.4 (Insight type, insight). An *insight type* is a name giving the semantics of an insight. Given a measure M, a categorical attribute B of a relation R, and two constants $val, val' \in Dom(B)$, an *insight* over R is a tuple $i = (M, B, val, val', p)$ where p is a selection predicate depending on the insight type.

In what follows we consider two types of insights: *mean greater* (M) and *variance greater* (V). The predicates associated with each type of insight are, respectively, $avg(val) > avg(val')$ (M) and $variance(val) > variance(val')$ (V).

Lemma 3.5 (Number of insights). *Let T be the number of insight types. The number of insights over R of schema $R[A_1, \ldots, A_n, M_1, \ldots, M_m]$ is*

$$\sum_{i=1}^{n} \binom{|dom(A_i)|}{2} \times m \times T$$

An insight induces a hypothesis over relation R.

Definition 3.6 (Hypothesis postulating an insight). Let R be a relation and $i = (M, B, val, val', p)$ be an insight over R. The hypothesis postulating i depends on the insight type: $E[X] > E[Y]$ (M) or $var(X) > var(Y)$ (V) where X and Y are the random variables over R representing measure M for predicates $B = val$ and $B = val'$, respectively.

The queries that express a comparison together with a given hypothesis are called *hypothesis queries*. An example is given in Figure 3, and they are defined as follows.

[2]In practice, as we will explain later, we use functional dependency detection in a pre-processing step to exclude meaningless queries, like selecting two days and grouping over months.

Table 1: Statistical tests by insight type

Insight type	Null hypothesis	Test statistics
M	$E[X] = E[Y]$	$\lvert\mu_X - \mu_Y\rvert$
V	$var(X) = var(Y)$	$\lvert\sigma_X^2 - \sigma_Y^2\rvert$

```
with comparison as
 (select t1.continent, April , May
  from
   (select month, continent, sum(cases) as April
    from covid where month = '4'
    group by month, continent) t1,
   (select month, continent, sum(cases) as May
    from covid where month = '5'
    group by month, continent) t2
  where t1.continent = t2.continent
  order by t1.continent)
select 'mean greater' as hypothesis from comparison
having avg(April)<avg(May);
```

Figure 3: A hypothesis query postulating insight $avg(April) < avg(May)$.

Definition 3.7 (Hypothesis query). Given a comparison query $q = (A, B, val, val', M, agg)$ and an insight $i = (M, B, val, val', p)$ of type τ, an *hypothesis query* is an extended relational query of the form:

$$\pi_{\tau \rightarrow hypothesis}(\sigma_p(q))$$

To be considered a true discovery, an insight has to be both (i) supported by the result of comparison queries and (ii) significant. For condition (i), the result of a comparison query is susceptible to trigger an insight only if it supports a hypothesis that can be tested.

Definition 3.8 (Query supporting an insight). Given a hypothesis query $h = \pi_{\tau \rightarrow hypothesis}(\sigma_p(q))$ for insight $i = (M, B, val, val', p)$ of type τ, h supports i, denoted $h \vdash i$, if h evaluates to true; consequently, q supports i for h, denoted $q \vdash_h i$, if $\sigma_p(q)$ is true. If $\sigma_p(q)$ is false, we say that h (resp., q) does not support i.

Note that a given comparison query q can support many insights. The set of insights supported by comparison query q is noted I_q in what follows. In what follows, we consider that the more insights supported by a query, the more interesting the query. Note also that an insight can be supported by many comparison queries. If we consider the set of insights of any type over R, for measure M, attribute B and values val, val', the set of comparison queries of the form $(A, B, val, val', M, agg)$ supporting such insights only differ in the grouping attribute A. In what follows, we consider that only the most interesting query from this set should be kept, since all the other queries would evidence the same insights.

As to condition (ii), the hypothesis postulating an insight corresponds to the alternative hypothesis of a statistical test for which the p-value indicates the significance of the insight. The test considered and the null hypothesis depend on the insight type (see Table 1).

Definition 3.9 (Insight significance). Let $\pi_{\tau \rightarrow hypothesis}(\sigma_p(q))$ be a hypothesis query for insight $i = (M, B, val, val', p)$ of type τ. The significance of i is $sig(i) = 1 - P(T > o|H_0)$, where o is the observed statistics over R, H_0 is the null hypothesis, and T is the random variable associated to the test results over R.

Example 3.10. An example of hypothesis query for the comparison query of Figure 2 is given in Figure 3. It postulates that the average number of cases for the month of April is less than the average number of cases for the month of May, i.e., insight $i = (cases, month, April, May, avg(April) < avg(May))$. The result of the comparison query of Figure 2 supports this, since it is observed at the continent level, $avg(May) - avg(April) = 61346.4$. To check the significance of the insight $avg(April) < avg(May)$, it is turned into the null hypothesis $E[X] = E[Y]$ where X and Y are the random variables representing cases for May and April, respectively. The test statistics $|\mu_X - \mu_Y|$ is applied over R, showing that $avg(May) - avg(April) = 55.79$. The p-value gives the significance of the insight as the probability to observe the test statistics value over R as extreme as the observation o. If the p-value is low enough, this means that the insight is both significant and supported by the comparison query, making this comparison query a good candidate for being presented to the user.

3.3 Insights and statistical errors

In statistical hypothesis testing, a type I error is the rejection of a true null hypothesis, while a type II error is the non-rejection of a false null hypothesis. Consistently with Zgraggen et al. [29], we associate false discoveries with type I error, a false discovery being in our case a query q supporting an insight i while the insight is not significant, i.e., $sig(i) < 0.95$ and $q \vdash i$. On the other hand, a false omission means ignoring a real pattern because it looks uninteresting, and corresponds to type II error. In our case, such a pattern corresponds to a query q not supporting an insight i while the insight is significant., i.e., $sig(i) > 0.95$ and $q \nvdash i$.

To quantify the evidence of an insight i, we define its credibility as the number of queries that support it.

Definition 3.11 (Credibility of an insight). Let i be an insight and Q^i be the set of hypothesis queries postulating i. The credibility of i is

$$credibility(i) = |\{h \in Q^i | h \vdash i\}|$$

For an insight i over schema $R[A_1, \ldots, A_n, M_1, \ldots, M_m]$, it is $|Q^i| = n - 1$.

The probability of making a type I error is a conditional probability, namely $\frac{credibility(i)}{|Q^i|}$ knowing that $sig(i) < 0.95$, while the probability of making a type II error is $1 - \frac{credibility(i)}{|Q^i|}$ knowing that $sig(i) \geq 0.95$. In what follows, we are interested only in significant insights, i.e., only those for which $sig(i) \geq 0.95$ is true.

Note that an insight, even if significant, may have no supportive hypothesis query, by construction of hypothesis queries. We choose not to consider this kind of insights, since no comparison seen by a user would trigger it.

For insight types like mean and variance, a transitivity relation allows to prune insights that can be deduced. If the mean of X is smaller than the mean of Y and the mean of Y is smaller than that of Z, then the mean of X is smaller than the mean of Z. In other words, the fact that the mean of X is smaller than that of Z is an insight that can be deduced from the other two, and can be pruned out from the set of insights. The same holds for variance.

4 COMPARISON NOTEBOOKS GENERATION

We now define the problem of generating sequences of comparison insights. Consistently with EDA (e.g., [11]), our objective is to generate compelling exploratory sessions, specifically, coherent sequences of comparison queries showing significant insights.

4.1 Problem formulation

Given a relation R and the set Q of all comparison queries over R, we are interested in producing a sequence of comparison queries from Q such that the sum of their execution costs is less than a time budget t, their total interestingness is maximal, and the overall distance between the queries is minimal.

This problem is defined formally in [5] as follows:

Definition 4.1 (Traveling Analyst Problem (TAP)). Let Q be a set of N queries, each associated with a positive time cost $cost(q_i)$ and a positive interestingness score $interest(q_i)$. Each pair of queries is associated with a metric $dist(q_i, q_j)$ representing the cognitive distance of browsing from one query result to the next. Given a time budget ϵ_t, the optimization problem consists in finding a sequence $\langle q_1, \ldots, q_M \rangle$ of queries, $q_i \in Q$, without repetition, with $M \leq N$, such that:

(1) $\max \sum_{i=1}^{M} interest(q_i)$
(2) $\sum_{i=1}^{M} cost(q_i) \leq \epsilon_t$
(3) $\min \sum_{i=1}^{M-1} dist(q_i, q_{i+1})$.

Lemma 4.2 (Complexity of TAP [5]). *TAP is strongly NP-hard.*

4.2 Interestingness, cost, and distance

Interestingness. Consistently with prior works [11, 16, 25, 29], our definition of comparison query interestingness is manifold: (i) the more insights supported, the better; (ii) the more significant the insights, the better; (iii) the more surprising the insights, the better; (iv) the more concise the comparison query, the better.

For (i), we just sum over the number of insights that a comparison query can support. For insight i, $sig(i)$ is used for (ii). As to (iii), we use the probability of the insight being a type II error. Finally, we use a conciseness measure in the spirit of that introduced in [11] for (iv).

Definition 4.3 (Interestingness of a query). Let q be a comparison query and I_q be the set of insights supported by q.

$$interest(q) = conciseness(\theta_q, \gamma_q) \times \sum_{i \in I_q} (\omega \times sig(i) \times (1 - \frac{credibility(i)}{|Q^i|}))$$

where

$$conciseness(\theta_q, \gamma_q) = e^{-\frac{1}{\theta_q^{\delta}}(\gamma_q - \theta_q \alpha)^2}$$

and ω is a weigh ruling the importance of $sig(i)$.

Conciseness uses a non-monotonic function of two variables: (i) θ_q, the number of tuples aggregated by query q, and (ii) γ_q, the number of groups in the result of q. Two values α and δ are used to control the tuple-to-group ratio. Parameter α sets the growth rate of the ideal number of groups given the number of tuples, behaving like the slope in a linear function. Parameter δ allows to "spread" the ideal ratio. A geometric intuition of the function behavior is given in Figure 4 (the undefined zone corresponds to the number of groups being greater than the number of tuples, which does not make sense in our context).

Distance. The distance we need should satisfy the triangle inequality since otherwise, given the problem formulation, the risk is to trade interestingness for distance. In other words, we could end up with sub-sequences where it is better to pass through a low-cost non-interesting query to reach an interesting one, while this would be impossible with a proper metric.

To keep the computation of this metric under control, we choose to use a weighted Hamming distance over the query parts. We recall that a comparison query q is represented as

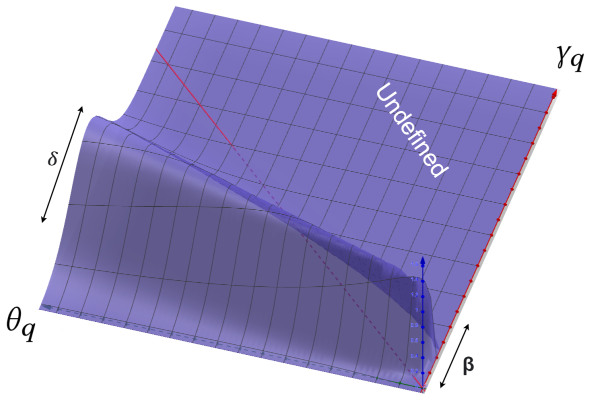


Figure 4: Illustration of the *conciseness* function

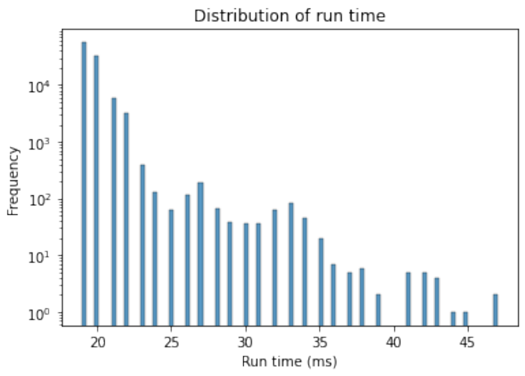


Figure 5: Distribution of comparison queries run times

a vector of query parts $q = (A, B, val, val', M, agg)$ that can be extracted from the text of the query. Consistently with prior work [2], the weights represent the importance of the query parts in the transition from one comparison query to another, precisely: val, val' the highest, followed by B, then A, and finally M and agg have the lowest impact.

Cost. The cost of a query should straightforwardly be its evaluation time. However, given the form of comparison queries, and assuming that no physical optimizations have been made, the cost of all comparison queries will roughly be the same. We ran a test with a sample of comparison queries over the ENEDIS dataset used in Section 6 which confirms this intuition (see Figure 5). In this case, only interestingness and distance should have an impact on the computation of an exact solution to the problem. In other words, we can set the cost of each query to the same value, and use the time budget for controlling the number of queries in the solution.

4.3 Generating the set of comparison queries

The generation of the set Q of comparison queries over a dataset R is given by Algorithm 1. The algorithm loops over all potential insights over R (lines 2-8), checking for each one its significance with the appropriate statistical test (line 3). If the insight is found to be significant (line 4), a comparison query is generated for it, by first generating a hypothesis query for each possible grouping attribute and aggregation function (line 5), then checking if this hypothesis query supports the insight (line 8). Finally, for the sets of comparison queries that evidence the same insights (line 15), only the ones maximizing interestingness are kept (line 16).

Algorithm 1 Comparison query generation

```
Require: a relation $R$
Ensure: a set of comparison queries over $R$
 1: $Q \leftarrow \emptyset$
 2: for each insight $i = (M, B, val, val', p)$ of type $\tau$ over $R$ do
 3:     compute $sig(i)$                          ▷ perform statistical test
 4:     if $sig(i) > 0.95$ then                    ▷ $i$ is significant
 5:         for each $A \neq B$ of $R$ and function $agg$ do
 6:             $q \leftarrow (A, B, val, val', M, agg)$
 7:             $h \leftarrow \pi_{\tau \rightarrow hypothesis}(\sigma_p(q))$
 8:             if $h \vdash i$ then                  ▷ $h$ supports $i$
 9:                 $Q \leftarrow Q \cup \{q\}$
10:             end if
11:         end for
12:     end if
13: end for
14: for each $q = (A, B, val, val', M, agg) \in Q$ do
15:     $Q^A \leftarrow \{q' \in Q | (C, B, val, val', M, agg)$ with $C \neq A\}$
16:     $Q \leftarrow Q \setminus Q^A \cup \{argmax_{q' \in Q^A} interest(q')\}$
17: end for
18: return $Q$
```

Algorithm 1 consists of a naive and inefficient approach for generating comparison queries. Computationally-wise, the costly steps are the statistical tests (line 3) and the evaluation of the hypothesis queries (line 8). We address these issues in the next section.

5 OPTIMIZING COMPARISON NOTEBOOK GENERATION

Generating comparison queries with Algorithm 1 and computing an exact solution of the TAP provides a basic approach to generate notebooks of comparison queries over small datasets. To scale to real-world datasets with very large number of insights and comparison queries, we implemented three types of optimizations: (i) optimizing statistical tests, (ii) minimizing the number of queries to send to the DBMS, and (iii) using a heuristic to approximate the TAP.

5.1 Optimizing statistical tests

5.1.1 Using permutation testing. For hypothesis testing we use resampling, due to its advantages over parametric testing [29]: it does not assume the distributions of the test statistics, nor does it impose samples to be large enough. We use the same permutations to check all possible insights on different measures for a given attribute, and correct the p-values using the Benjamini-Hochberg FDR correction [3].

5.1.2 Using sampling. We use two different offline sampling strategies to speed-up the statistical tests. The first one, *unbalanced-sampling*, samples each of the n categorical attributes independently. It seeks to balance the number of tuples per attribute value, avoiding that very selective values be under-represented. The second one, *random-sampling*, randomly samples the dataset in a uniform way.

5.2 Reducing the number of queries

Algorithm 1 requires (i) for doing statistical tests, to evaluate n queries of the form $\pi_{A,M}(R)$, where A is a categorical attribute in $\{A_1, \ldots, A_n\}$ and M is a measure attribute in $\{M_1, \ldots, M_m\}$, and (ii) for generating comparison queries, to evaluate all hypothesis queries for significant insights, i.e., in the worst case all hypothesis queries.

Our aim is to reduce the number of queries to send to the DBMS by finding a set of queries retrieving all necessary data for the statistical tests and the generation of comparison queries. We first remark that we should separate the computation of statistical tests from the evaluation of hypothesis queries because doing both at the same time would require to evaluate $n(n-1)/2$ queries of the form $\pi_{A,B,M_1,\ldots,M_m}(R)$ over the full dataset, which for large instances is almost as large as the instance itself.

5.2.1 Bounding the number of queries. To reduce the number of queries for doing the statistical tests, we send n queries of the form $\pi_{A,M_1,\ldots,M_m}(R)$. To reduce the number of hypothesis queries to evaluate, we remark that we need only all the group-by sets of two categorical attributes (named 2-group-by sets from now on) taken in a given order. This corresponds to $n(n-1)/2$ queries (see Lemma 3.2) of the form $\gamma_{A,B,agg_1(M_1),agg_1(M_2),\ldots,agg_f(M_m)}(R)$ where A, B are two different categorical attributes from the schema $R[A_1, \ldots, A_n, M_1, \ldots, M_m]$, and the agg_i are all the aggregate functions. This also provides an upper bound to the number of queries to launch to retrieve the data necessary to evaluate the set of hypothesis queries.

5.2.2 Merging group-by queries. To further reduce the number of hypothesis queries, we use a group-by aggregate merging strategy similar to the one used in [15] and presented in Algorithm 2. Specifically, we look for the largest group-by sets fitting in memory from which many hypothesis queries can be evaluated, and evaluate them for free once the data they need is in memory. The problem of finding the best set of group-by sets is an instance of the classical weighted set cover problem. Let R be a relation over the set $A = \{A_1, \ldots, A_n\}$ of n categorical attributes. Let G be the set of all group-by sets from R except the 1-group-by sets, i.e., $G = 2^A \setminus \{A_1\} \setminus \ldots \setminus \{A_n\}$ (line 2). Assume that we have a weight for each elements of G corresponding to their estimated memory footprint, as obtained from the query optimizer (line 6). The goal is to find the sub-collection of G having the minimal overall weight that covers the set U of 2-group-by sets. This problem being NP-hard, we use a greedy heuristic to approximate the solution to the weighted set cover problem (line 8), whose complexity is $O(|U| \times log|G|)$ [28]. In case the smallest subset of aggregates does not fit in memory, we implement a fallback strategy that successively loads the smallest possible aggregates (i.e., the group by sets of U) in main memory.

5.3 Solving the TAP

Solving the TAP problem (see Section 4) exactly is done with a mathematical model on CPLEX 20.10 and is implemented in C++[3]. For large datasets with millions of queries, finding exact solutions or even using complex polynomial heuristic is intractable. We use a fast and memory-efficient heuristic inspired by the classic "sort by item efficiency" heuristic for solving the Knapsack problem [9], presented in Algorithm 3. The trick is to sort all queries of Q in decreasing order of interest/cost (lines 1-4), constructing the solution by iterating over the sorted Q (lines 7-14), and adding

[3]https://github.com/AlexChanson/Cplex-TAP

Algorithm 2 Finding the best set of group-by sets

```
Require: a relation $R$ with $n$ categorical attributes $A = \{A_1, \dots, A_n\}$
Ensure: a set of group-by sets over $R$ with minimal memory footprint covering all pairs of categorical attributes
1: $G \leftarrow 2^A$
2: $G \leftarrow G \setminus \{g \in G || g| = 1\}$
3: $U \leftarrow \{g \in G || g| = 2\}$
4: for each group-by set $g$ of $G$ do
5:     $q \leftarrow \gamma_g(R)$
6:     Estimate the size of $q$
7: end for
8: $G \leftarrow$ Solve the weighted set cover problem for $G, U$
9: return $G$
```

Table 2: Description of the datasets

Name	Size (tuples)	Size (Bytes)	#Categ. attr.	Adom size (min-max)	#Meas.	#Comp. queries
Vaccine	5045	656K	6	2-107	1	700
ENEDIS	114,527	21M	7	3-1295	2	1,571,832
Flights	5,819,079	808M	5	7-377	3	350,460

a query to the sequence if (i) the budget is not spent (line 10), and (ii) the global distance of the sequence is not over a pre-fixed bound ϵ_d (line 10). The query is then inserted in a position of the sequence that minimizes the global distance (line 11). Assuming the size of the solution is much smaller than $|Q|$, which depends on the bounds set by ϵ_d and ϵ_t, the most costly operation is the sort (line 4), which can be achieved in the worst case in $O(|Q| \times log(|Q|))$ time. This stands for any practical purpose, especially for large datasets that will yield hundreds of thousands of insights while the notebook sizes, controlled with ϵ_t, are expected to remain readable and thus much shorter. Additionally, distances can be computed on the fly, limiting memory consumption.

While this heuristic can be applied to the TAP as defined above, we recall from Section 4 that all comparison query costs can be set to the same value, further simplifying Algorithm 3: Q can be sorted based only on interest, and ϵ_t is used to bound the number of queries in the solution.

As classically done for simplifying the resolution of such multi-objective problems, both exactly and for heuristic approaches, one objective is transformed into a constraint. In the case of the TAP, the distance objective is transformed into $\sum_{i=1}^{M-1} dist(q_i, q_{i+1}) < \epsilon_d$. Varying ϵ_d allows to generate different points on the Pareto front of the original multi-objective problem [26]. For instance, smaller values of ϵ_d force solutions with queries that are closer to each other. The complete mathematical model used in CPLEX is described in [7].

6 EXPERIMENTAL RESULTS

6.1 Experimental setup

The real datasets for our tests are described in Table 2. The tiny Vaccine dataset[4] consists of country-level Covid19 vaccination data as of June 2021. The ENEDIS dataset[5] is about electric consumption in France by location, year, consumption category, and commercial sector. The Flights dataset[6] consists of one year of flight arrival and departure details for all commercial flights within the USA.

Algorithm 3 Adaptation of the "sort by item efficiency" heuristic

```
Require: A set of queries $Q$ with their $cost$ and $interest$ two reals $\epsilon_t$ (time budget), $\epsilon_d$
Ensure: an approximate solution to the TAP, of size at most $\epsilon_t$
1: for $q \in Q$ do
2:     $weight(q) \leftarrow interest(q)/cost(q)$
3: end for
4: $\mathfrak{Q} \leftarrow$ sort $Q$ by weights in decreasing order
5: $t \leftarrow 0$
6: $S \leftarrow []$
7: for $q \in \mathfrak{Q}$ do
8:     $\mathfrak{S} \leftarrow$ the set of possible unique inserts of $q$ in $S$
9:     $min_d \leftarrow min_{\mathfrak{S}}(\sum_{i=1}^{|S|} distance(q_i, q_{i+1}))$
10:    if $t + cost(q) < \epsilon_t$ and $min_d < \epsilon_d$ then
11:        $S \leftarrow argmin_{\mathfrak{S}}(\sum_{i=1}^{|S|} distance(q_i, q_{i+1}))$
12:        $t \leftarrow t + cost(q)$
13:    end if
14: end for
15: return $S$
```

Table 3: Implementations

Name	Generation of Q	Solving TAP
Naive-exact	Algo. 1 + bounding	CPLEX
Naive-approx	Algo. 1 + bounding	Algo. 3
WSC-approx	Algo. 2	Algo. 3
WSC-unb-approx	Algo. 2 + unbalanced-sampling	Algo. 3
WSC-rand-approx	Algo. 2 + random-sampling	Algo. 3

The implementations used in the tests are described in Table 3, where we detail the algorithm for generating the set of comparison queries and the algorithm for solving the TAP, using naive implementations and the optimizations presented in Section 5. Naive-exact uses the naive Algorithm 1 and the optimization of Section 5.2.1 for generating the set of comparison queries, and then vanilla CPLEX for solving the TAP. In Naive-approx, approximating TAP is done with our adaptation of the sort by item efficiency heuristics described in Algorithm 3. This heuristics is used for implementations of WSC-approx, WSC-unb-approx, and WSC-rand-approx, which use Algorithm 2 of Section 5.2.2 to reduce the number of queries and differ in how statistical tests are done: no sampling for WSC-approx, unbalanced sampling for WSC-unb-approx, and random-sampling for WSC-rand-approx. In all our tests, the parameters of the conciseness function (see Section 4) are set to values empirically tuned to a good trade-off between the number of groups and the number of tuples aggregated, and ϵ_d is set to a value empirically tuned to obtain TAP solutions where queries are very close to each other.

Our prototype is written in Java and is publicly available[7]. It runs on top of PostgreSQL version 13.4. We implemented a pre-processing step to detect functional dependencies among categorical attributes, to prevent meaningless queries from being generated. All tests were run on a Fedora Linux (kernel 5.11.13-200) workstation, on a 2.3 Hz Intel Xeon 5118 12-core, 24 logical processors and 377GB 2666 MHz of DDR4 main memory.

[4]https://www.kaggle.com/gpreda/covid-world-vaccination-progress
[5]https://data.enedis.fr/explore/dataset/consommation-electrique-par-secteur-dactivite-iris/export/
[6]https://community.amstat.org/jointscsg-section/dataexpo/dataexpo2009

Table 4: Time to solve the TAP to optimality

#Queries	Time (s)				%Timeouts
	avg	min	max	stdev	
100	1.61	0.65	8.14	1.62	0
200	28.47	3.12	126.92	31.52	0
300	239.83	12.28	963.55	240.12	0
400	727.90	24.47	1667.2	414.51	0
500	1869.75	166.15	> 3600	830.74	23.3
600	1343.89	240.06	> 3600	1000.37	86.7
700	-	> 3600	> 3600	-	100

6.2 Exact resolution of the TAP

This first test aims at answering the following question: how many queries can be reasonably handled when computing the exact solution to the TAP? For this test, we generated artificial sets of queries of different sizes, from 100 to 700 queries (reaching the size of the set of comparison queries of our smallest dataset, Vaccine), varying the number of comparison queries, while keeping similar uniform distributions of interestingness, cost, and distances. We ran vanilla CPLEX with defaults settings (notably single-threaded), with a timeout set to one hour, on 30 instances of equal size, for a given number of queries ($\epsilon_t = 25$) in the solution. We report the average time by size in Table 4.

Timeouts are reached from 500 queries onward; when reaching the size of our smallest dataset (700 queries) the solver always took more than one hour, preventing the calculation of average and standard deviation. This instance size is then ignored in subsequent tests. The fact that the average time is lower for 600 queries (compared to 500) is explained by the high number of timeouts for this instance size, which are ignored in the computation of the average. This shows that, expectedly, the TAP cannot be solved exactly for large datasets, so heuristics will be used in the following. Note that assessing the quality of the approximate solutions found heuristically will be done by comparing them to the exact solutions found on our smallest dataset.

6.3 Scalability

Since, as shown above, exact resolution is practically possible only for small datasets, this next test aims at answering the question: how well does the implementation of a reasonable approximate solution to the problem scale? To answer this question, we ran many tests to check the different optimizations presented in the previous section.

For each implementation, we show the time to compute a notebook, broken down into generation and solving time, varying the dataset size and the number of queries expected in the solution, i.e., ϵ_t (see Section 4), called budget in what follows.

We start by adjusting the sample size for the two implementations that use sampling: WSC-unb-approx and WSC-rand-approx.

6.3.1 Adjusting sample size. This test aims at finding what sample size to use on large datasets, to achieve a good compromise between runtime and percentage of insights detected. We ran WSC-unb-approx and WSC-rand-approx on the Enedis dataset, varying the sample size and reporting the runtime and the fraction of insights found. As shown in Figure 6, 20% seems a good compromise for WSC-unb-approx while WSC-rand-approx needs larger samples, around 40%, to achieve a similar ratio of insights that can be detected. This is mainly due to the ability of unbalanced sampling to better preserve the initial dataset diversity, particularly minority trends, which in turn helps preserving more insights at lower sampling rates.

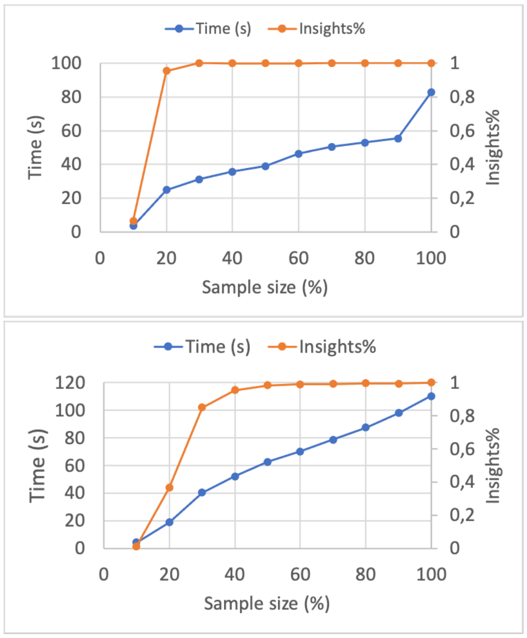


Figure 6: Adjusting sample size for WSC-unb-approx (top) and WSC-rand-approx (bottom)

6.3.2 Runtime by budget. For this test, we ran the 5 implementations on the Enedis dataset, varying ϵ_t, i.e., the number of comparison queries in the solution, in $\{5, 10, 20, 40\}$.

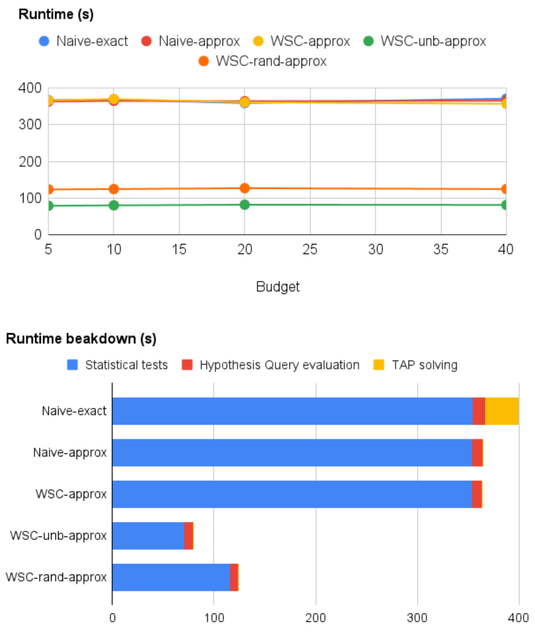


Figure 7: Runtimes by budget ϵ_t (top) and breakdown (bottom) on the ENEDIS dataset

The result is shown in Figure 7, which shows the runtime by budget (top) and the average runtime breakdown (bottom).

[7]https://github.com/patrickmarcel/sqlEDAqueryGenerator

Importantly, for Naive-exact, the TAP resolution timed out and is not counted, meaning that the runtime shown in Figure 7 (top) is only for the generation of Q. It can be observed that the runtime is quite steady for all values of ϵ_t, for all implementations. This confirms the remark made in Section 5.3 that, for approximate solutions, the value of ϵ_t has no impact when it is much smaller than $|Q|$.

As expected, the implementations using sampling strategies outperform the others, i.e., naive-exact, naive-approx, and WSC-approx, which are all between 300 and 400 seconds. The sample sizes were adjusted based on the observations reported above, and therefore WSC-rand-approx runs on a larger sample, which explains why WSC-unb-approx, while using a more sophisticated sampling strategy, runs faster. However, even with this larger sample, only 85% of insights can be tested on average by WSC-rand-approx, compared to 95% for WSC-unb-approx. As to the breakdown for the different steps of the implementations, we observe that performing the statistical tests is the most costly step, with sampling drastically reducing it. Solving TAP is negligible, except obviously for the exact resolution. Algorithm 2 has only little impact on the hypothesis query evaluation, which can be explained by the small number of categorical attributes in the dataset.

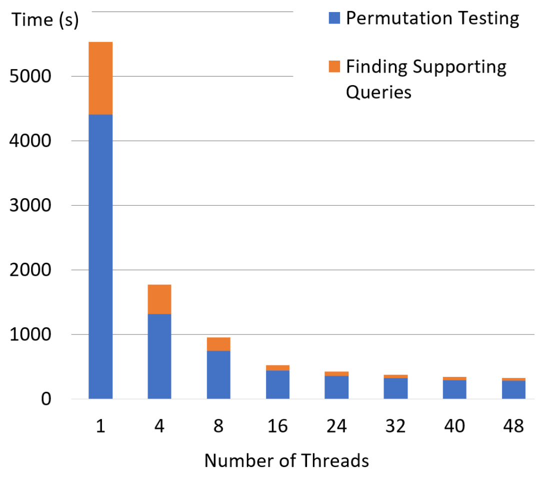


Figure 8: Impact of multi-threading on the generation of Q

6.3.3 Multi-threading. Several steps of the generation of Q can be parallelized, notably (i) permutation testing over different groups of categorical attributes and (ii) the use of in-memory partial aggregates to check which comparison queries support the insights. We run WSC-approx on the ENEDIS dataset, with 1 to 48 threads and report the runtime for steps (i) and (ii) in Figure 8. The speedup from single threaded to only 8 threads is very large, and remains substantial when going from 8 to 16 threads. However, further increasing the number of threads yields diminishing returns. The main reason for this is related to the architecture of the processor used for our tests, which produces overhead when increasing the number of threads over 24. In our tests, we therefore set the number of threads to 16.

6.3.4 Runtime of sampling strategies on larger datasets. Running WSC-approx on the Flights dataset took more than 14 hours. To be able to generate comparison notebooks more efficiently for this dataset, we ran WSC-unb-approx and WSC-rand-approx on Flights, testing different sample sizes in {5%, 10%, 20%, 30%}. The results are shown in Figure 9.

Table 5: Average deviation to optimal solution objective

#Queries	Deviation
100	1.14 ±1.52 %
200	0.17 ±0.12 %
300	0.10 ±0.09 %
400	0.06 ±0.06 %
500	0.06 ±0.05 %
600	0.03 ±0.04 %

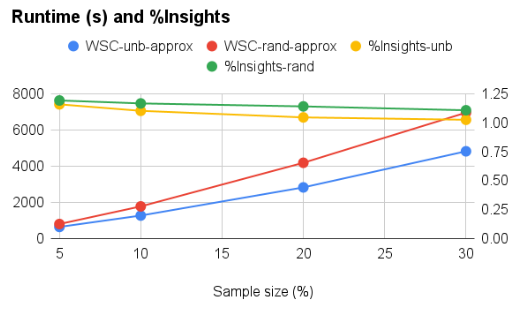


Figure 9: Runtime and % of insights on the Flights dataset

It can be seen that WSC-unb-approx outperforms WSC-rand-approx, as already observed on the ENEDIS dataset. Analyzing the runtime breakdown, we see that the last two steps remain insensitive to the sample size, around 20 seconds for Hypothesis query evaluation and around 300 milliseconds for TAP solving. Note that the percentage of insights detected, for both implementations, is greater than 1. This is due to the extreme reduction in the dataset size using aggressive sampling factors during statistical tests: some insights detected are spurious, and the quantity of spurious insights decreases as the sampling factor increases. We observe that WSC-unb-approx uses a sampling strategy that is more robust to the spurious insights than that of WSC-rand-approx. Tuning the credibility component of our interestingness function (see Section 4.2), being computed on the complete dataset, could be used to control the weight given to these spurious insights.

6.4 Quality of approximate solutions

This test aims at answering the question: how degraded are the approximate solutions of the TAP compared to the exact ones? For this test, we used Algorithm 3 to find approximate solutions of the TAP.

Our first experiment uses the same artificial datasets as the ones used in Section 6.2, with the same protocol (averaging the results over 30 runs on instances of equal size, fixing the size of the solutions ϵ_t). As a measure of quality of the solution, we compute z, i.e., the sum of interestingness of the queries in the solutions. We show in Table 5 $((cplex.z - algo3.z)/cplex.z) \times 100$, i.e., the deviation between $cplex.z$, the quality of solutions found with CPLEX, and $algo3.z$, the quality of solutions found by Algorithm 3.

The deviation remains very low in general, indicating that the heuristics used by Algorithm 3 is effective when considering

Table 6: Deviation to optimal solution

#Queries	Recall (Algorithm 3)	Recall (Baseline)
100	0.285 ± 0.085	0.122 ± 0.062
200	0.296 ± 0.054	0.089 ± 0.038
300	0.270 ± 0.041	0.094 ± 0.028
400	0.285 ± 0.033	0.087 ± 0.021
500	0.285 ± 0.027	0.094 ± 0.024
600	0.279 ± 0.032	0.095 ± 0.017

Table 7: Notebook generators for user tests

Name	Sampling	Sample size	Interestingness	Solving TAP
Naive-exact	-	100%	full	CPLEX
WSC-approx	-	100%	full	Algo. 3
WSC-approx-sig	-	100%	sig. only	Algo. 3
WSC-approx-sig-cred	-	100%	sig. and cred. only	Algo. 3
WSC-unb-approx	unbalanced	10%	full	Algo. 3
WSC-rand-approx	random	10%	full	Algo. 3

query interestingness. Deviations are greater for small instances and decrease with larger sizes. This is expected since, for smaller instances, as distribution of interestingness is uniform and the size of solutions is fixed, the probability of picking an uninteresting query is higher.

Our second experiment consists in measuring the recall of comparison queries in the solution, i.e., the proportion of queries present in the optimal solution that are found by the heuristic. We use the same protocol as above, and show the average recall in Table 6.

We observe that the heuristic manages to find around 30% of the queries present in the optimal solution, on average, varying very little with the instance size. While this recall appears relatively low, it is counterbalanced by the fact that the heuristic picks queries with high interestingness for the remaining 70% of the solution, as illustrated by the deviations in Table 6. We implemented a baseline consisting of picking the top ϵ_t queries in terms of interestingness, and compared the recall of this Baseline to that of Algorithm 3. As shown in Table 6, Algorithm 3 is steadily around 2.5 to 3 times better than the Baseline.

6.5 Human evaluation

This last test aims at answering the questions: which version of the notebook generator is favored by users? Are the notebooks generated using sampling or approximating the TAP also approved by users?

For this test, we recruited 9 volunteer PhD students or lecturers from France and Italy with at least some basic knowledge in data science. We generated a collection of 6 notebooks of 10 comparison queries each on the ENEDIS dataset, using different versions of our generator detailed in Table 7. The versions of Naive-exact, WSC-approx, WSC-unb-approx, and WSC-rand-approx are as described in Table 3. In addition, we used two more versions of WSC-approx: WSC-approx-sig is WSC-approx where the interstingness score of the comparison queries is computed with only the significance of the insights, i.e., without conciseness nor credibility, while WSC-approx-sig-cred is WSC-approx where the interstingness score of the comparison queries is computed with the significance and credibility, but without conciseness.

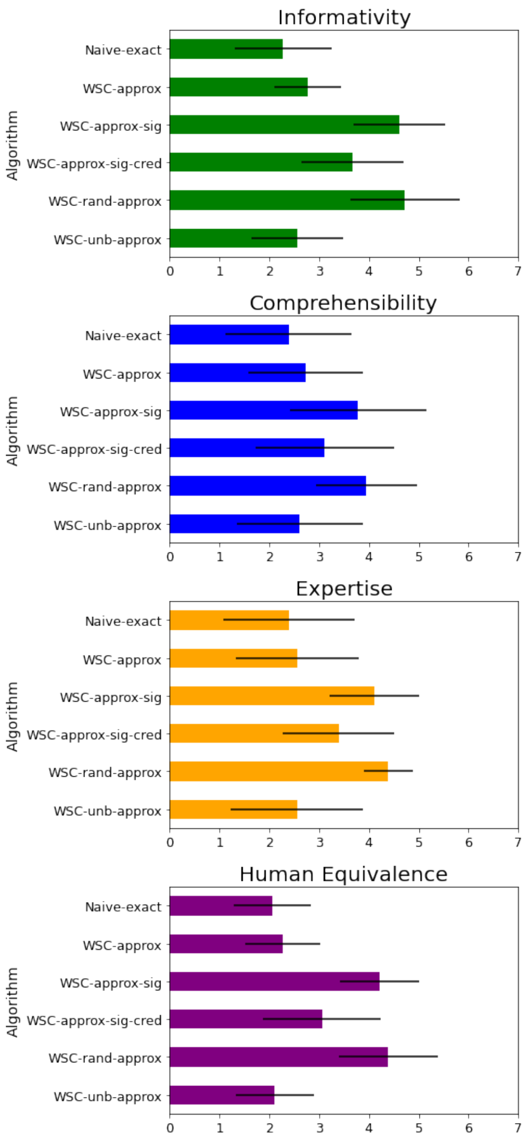


Figure 10: Qualitative human evaluation

The notebooks generated were deployed on Jupyter[8] and were presented to the volunteers, insisting on the fact that the notebooks should be considered as starting points of the exploration of a potentially unknown dataset. We also provided a brief data dictionary explaining some business terms of the ENEDIS dataset. We asked them to rate the notebooks, on a scale from 1 (lowest) to 7 (highest), using the 4 criteria proposed in [11]: (1) Informativity — How informative is the notebook and how well does it capture dataset highlights? (2) Comprehensibility — To what degree is the notebook comprehensible and easy to follow? (3) Expertise — What is the level of expertise of the notebook composer? (4) Human Equivalence — How closely does the notebook resemble a human-generated session?

We show the average scores given by testers in Figure 10. In general, the main observations are that WSC-rand-approx and SC-approx-sig dominates the other on all criteria, while Naive-exact is dominated on all criteria. The fact that WSC-rand-approx

[8]https://datastory.lifat.fr/tree?#notebooks, Password: 1598Rksil%42TAP

obtains the best scores indicates that sampling does not seem to systematically affect how users consider the insights in the notebooks. In particular, even if some insights may be missed by the approach, as explained above, the notebook generated can still be deemed informative. The low scores received by Naive-exact tend to indicate that an exact resolution is not needed for a notebook to be well perceived by users. As to the interestingness measure, the test is inconclusive in ruling out one or the other of the components, from a user's perspective. We also recall that notebooks were generated with values of ϵ_d favoring solutions where comparison queries are very close to each other, which may have disappointed users preferring more diversity in the notebook, and might explain the low scores on the Human equivalence criterion. Interestingly, a statistical t-test confirmed that the difference in the positive evaluations received by WSC-rand-approx and SC-approx-sig is not significant. Moreover, this difference is not significant either on the comprehensibility criteria with WSC-approx-sig-cred (even if in this case the p-value is around 0.16 which indicates a weaker confidence in the conclusion). Another interesting observation concerns Naive-exact, which is supposedly the optimal approach. However, human evaluations rather show that it is overall the least appreciated method. This is nuanced by the following elements: (i) recall studies as presented in Table 6 show that in general around 30% of the queries in Naive-exact and the other approaches are actually the same, and (ii) t-tests on human evaluation criteria results show that there are no significant differences between Naive-exact and WSC-approx or WSC-unb-app approaches. The latter point confirms the previous observations: firstly, our heuristic is perceived similarly as an exact resolution, and second, it cannot be said that sampling affects how human evaluate the notebooks.

7 CONCLUSION

This paper addressed the problem of generating SQL notebooks of comparison queries to support Exploratory Data Analysis. We introduced the definitions of comparison insights, hypothesis queries, comparison queries, and comparison query interestingness, and formalized the problem of generating notebooks of comparison queries that are insightful and coherent. We presented approaches for computing exact and approximate solutions to this problem, and ran experiments on artificial and real datasets to understand the scalability of the implementations and the accuracy of approximate solutions.

Our future work includes the tuning of our notebooks generators, especially to speed up the generation of the set of comparison queries and to avoid the generation of spurious insights on large datasets. We also aim at characterizing the forms of insights that can be included in our approach. Indeed, although the present work is restricted to comparison insights, our approach can be extended to other forms of insights. This requires to (i) find an appropriate SQL hypothesis query that expresses the insight, (ii) find an appropriate statistical test for it, and (iii) adapt the interestingness, distance, and cost functions. Finally, we plan to extend our approach to other forms of popular analytical queries (like, e.g., explain queries [1]), and to the possibility of mixing queries and model computation.

REFERENCES

[1] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Anathanaraya, John Sheu, Erik Meijer, Xi Wu, Jeffrey F. Naughton, Peter Bailis, and Matei Zaharia. DIFF: A relational interface for large-scale data explanation. *Proceedings of VLDB Endow.*, 12(4):419–432, 2018.

[2] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.*, 39(2):463–489, 2014.

[3] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate - a practical and powerful approach to multiple testing. *J. Royal Statist. Soc., Series B*, 57:289 – 300, 11 1995.

[4] Tom Blount, Laura Koesten, Yuchen Zhao, and Elena Simperl. Understanding the use of narrative patterns by novice data storytellers. In *Proceedings of CHIRA*, pages 128–138, Budapest, Hungary, 2020.

[5] Alexandre Chanson, Ben Crulis, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Stefano Rizzi, and Panos Vassiliadis. The Traveling Analyst Problem: Definition and preliminary study. In *Proceedings of DOLAP@EDBT/ICDT*, pages 94–98, Copenhagen, Denmark, 2020.

[6] Alexandre Chanson, Thomas Devogele, Nicolas Labroche, Patrick Marcel, Nicolas Ringuet, and Vincent T'kindt. A chain composite item recommender for lifelong pathways. In *Proceedings of DaWaK*, pages 55–66, 2021.

[7] Alexandre Chanson, Nicolas Labroche, Patrick Marcel, and Vincent T'Kindt. The Traveling Analyst Problem, Orienteering applied to exploratory data analysis. In *Proceedings of ROADEF*, 2021.

[8] Conor Cunningham, Goetz Graefe, and César A. Galindo-Legaria. PIVOT and UNPIVOT: optimization and execution strategies in an RDBMS. In *(e)Proceedings of VLDB*, pages 998–1009, Toronto, Canada, 2004.

[9] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957.

[10] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. QuickInsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of SIGMOD*, pages 317–332, Amsterdam, The Netherlands, 2019.

[11] Ori Bar El, Tova Milo, and Amit Somech. Automatically generating data exploration sessions using deep reinforcement learning. In *Proceedings of SIGMOD*, pages 1527–1537, Portland, OR, USA, 2020.

[12] Matteo Francia, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Panos Vassiliadis. Assess queries for interactive analysis of data cubes. In *Proceedings of EDBT*, pages 121–132, Nicosia, Cyprus, 2021.

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

[14] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of SIGMOD*, pages 277–281, 2015.

[15] Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. MetaInsight: Automatic discovery of structured knowledge for exploratory data analysis. In *Proceedings of SIGMOD*, pages 1262–1274, 2021.

[16] Patrick Marcel, Verónika Peralta, and Panos Vassiliadis. A framework for learning cell interestingness from cube explorations. In *Proceedings of ADBIS*, pages 425–440, 2019.

[17] Tova Milo and Amit Somech. Next-step suggestions for modern interactive data analysis platforms. In *SIGKDD*, pages 576–585. ACM, 2018.

[18] Julian Ramos Rojas, Mary Beth Kery, Stephanie Rosenthal, and Anind K. Dey. Sampling techniques to improve big data exploration. In *Proceedings of LDAV*, pages 26–35, Phoenix, AZ, USA, 2017.

[19] Senjuti Basu Roy, Gautam Das, Sihem Amer-Yahia, and Cong Yu. Interactive itinerary planning. In *Proceedings of ICDE*, pages 15–26, 2011.

[20] Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *Proceedings of VLDB*, pages 42–53, 1999.

[21] Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *Proceedings of VLDB*, pages 307–316, 2000.

[22] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proceedings of EDBT*, pages 168–182, 1998.

[23] Gayatri Sathe and Sunita Sarawagi. Intelligent rollups in multidimensional OLAP data. In *Proceedings of VLDB*, pages 531–540, 2001.

[24] Tarique Siddiqui, Surajit Chaudhuri, and Vivek R. Narasayya. COMPARE: accelerating groupwise comparison in relational databases for data analytics. *Proceedings of VLDB Endow.*, 14(11):2419–2431, 2021.

[25] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of SIGMOD*, pages 1509–1524, Chicago, IL, USA, 2017.

[26] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms.* Springer, 2006.

[27] Pieter Vansteenwegen and Aldy Gunawan. *Orienteering Problems.* EURO Advanced Tutorials on Operational Research. Springer, 2019.

[28] Neal E. Young. Greedy set-cover algorithms. In *Encyclopedia of Algorithms*, pages 886–889. Springer, 2016.

[29] Emanuel Zgraggen, Zheguang Zhao, Robert C. Zeleznik, and Tim Kraska. Investigating the effect of the multiple comparisons problem in visual analysis. In *Proceedings of CHI*, page 479, Montreal, QC, Canada, 2018.