

# The Traveling Analyst Problem: Definition and preliminary study

Alexandre Chanson, Ben  
Crulis, Nicolas Labroche,  
Patrick Marcel, Verónika  
Peralta

University of Tours, France  
firstname.lastname@univ-tours.fr

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

Panos Vassiliadis  
University of Ioannina, Greece  
pvassil@cs.uoi.gr

## ABSTRACT

This paper introduces the Traveling Analyst Problem (TAP), an original strongly NP-hard problem where an automated algorithm assists an analyst to explore a dataset, by suggesting the most interesting and coherent set of queries that are estimated to be completed under a time constraint. We motivate the problem, study its complexity, propose a simple heuristic under simplifying assumptions for approximating it, and run preliminary tests to observe the behavior of this heuristic.

## 1 INTRODUCTION

Interactive data analysis (IDE) [10, 19] is an iterative process consisting in executing an action (e.g., a query or a pattern extraction algorithm) over the data, receiving the result and deciding what query comes next. It is a challenging task that a number of previous works aimed at facilitating (see e.g., [5, 19]). Automating such a process raises a number of challenges [16, 25]: how to determine the direction to follow in often very large and disorienting datasets, how to decide what is the best query to apply, how to determine if a result is interesting, how to tell a story with the data resulting from the analysis [8, 9], etc.

If we define a *data story* as a coherent sequence of queries that answer a user goal, we can express this problem as the computation of the most interesting and coherent data story that can be obtained within a reasonable time. Even with simplifying assumptions, like restricting to exploratory OLAP queries over a multidimensional schema (e.g., a star-schema, which allows navigating hierarchically-structured data with a low formulation effort) and giving a particular starting point, this problem remains inherently highly combinatorial.

This paper presents a preliminary study of this problem, that we name the *Traveling Analyst Problem* (TAP). Similarly to automated machine learning, which aims at finding the best model on a dataset given a time budget (see e.g., [7]), TAP aims at (i) finding, from a very large set of candidate queries, a subset of queries that maximizes their interest within a limited time budget, and (ii) ordering them so that they narrate a coherent data story. More formally, each query is associated to an interest score as well as to an execution cost. A distance between queries is used to order the queries so that the transition cost between two consecutive queries is minimized. Interestingly, a study of the state-of-the-art reveals that TAP has not been studied in the Operations Research community, while being close to two classical optimization problems (the Traveling Salesman Problem and the Knapsack Problem) [11].

Practically, we envision TAP as being at the core of an IDE system being used by data analysts, like the one described in [16]. Analysts would not directly enter TAP parameters, but would use storytelling mechanism instead (e.g., [8]) or patterns like “I want a quick short report around this query result” or “I want an in-depth analysis in this particular zone of the cube”.

The contributions of this preliminary effort include the formalization of TAP as well as the proof of its strong NP-hardness (Section 2), an approximation algorithm (Section 3), and some preliminary tests assessing the parameter sensitivity, execution time, and closeness to the optimal of our heuristics (Section 4). Section 5 concludes the paper by discussing future research.

## 2 PROBLEM FORMULATION AND COMPLEXITY

In this section, we formalize TAP and show its NP-hardness.

### 2.1 Problem formulation

TAP is formulated as:

**Input:** Given a set of  $n$  queries, a function *interest* estimating an interestingness score for each query, a function *cost* estimating the execution cost of each query, and a function *dist* estimating a cognitive distance between queries,

**Do:** find a sequence of  $m \leq n$  queries (without repetition)

**S.T.:** the sequence has the following properties:

- (1) it maximizes the overall interestingness score,
- (2) the sum of the costs does not exceed a user-specified time budget  $t$ ,
- (3) it minimizes the overall cognitive distance between the queries.

We assume that a storytelling mechanism (e.g., [8]) generates a set of candidate queries towards producing a story. Thus, our deliberations start with a set of  $n$  candidate queries whose execution has to be optimized. Formally, 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  $dist(q_i, q_j)$  for their cognitive distance. Given a time budget  $t$ , the optimization problem consists in finding a sequence  $\langle q_1, \dots, q_m \rangle$  of queries,  $q_i \in Q$ , without repetition, with  $m \leq n$ , such that:

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

The decision problem associated with this optimization problem is to decide if such a sequence exists.

## 2.2 Complexity

TAP can be related to two families of classical NP-hard optimization problems: (i) the Knapsack problem, which consists in picking weighted items from a set  $S$  such that the sum of their values is maximum, without exceeding a given size [13], corresponding to constraints (1) and (2); and (ii) the traveling salesman problem (TSP), which aims at finding the shortest route that visits all cities and returns to the initial one [1] and is close to constraint (3).

In our context, the former problem would find the most interesting queries given a time budget but, in its classical formulation, it would miss the ordering of queries. A variant [4] includes the position of each object in the Knapsack via the definition of a function (which is constant in the case of classical Knapsack). While this problem is closer to TAP, it involves a function that only relies on the position (or order) of an object in the Knapsack, and not on the positions of objects previously picked.

TAP could also be modeled as a TSP problem with particular constraints: (i) the TSP cities are the queries; (ii) inter-city distances correspond to the cognitive distance between queries, whose total must be minimized. However, differently from classical TSP, TAP operates under strong constraints: (iii) it is possible to visit only a subset of cities, each city has a weight corresponding to the action's interest, whose total is to be maximized; and (iv) each city has a duration of visit corresponding to the cost of the query, whose sum must not go beyond a given time budget.

Interestingly, this optimization problem has not been studied in the literature yet. The variant of the TSP called *TSP with profit* (TSPwP), described by Feillet & al. in [6], is closer to our problem, but still differs in two aspects: (i) it looks for circuits and does not reject any vertex in the solution, and (ii) it gives a limit in terms of the travel distance (inter-queries distance in our case) while our limit is on the cost of queries (the duration of visit).

An in-depth study of the TAP complexity is beyond the scope of this paper. However, we can easily show that our problem is strongly NP-hard since the TSP is a particular case of it. Indeed, if the time budget  $t$  is high enough, i.e., all queries can be selected, then TAP is a TSP. This result means that, unless  $P=NP$ , the TAP problem can only be solved to optimality by algorithms with a worst-case time complexity in  $O^*(c^n)$ , with  $c$  a positive root and  $n$  the size of  $Q$ .

## 2.3 Size of the problem and our naive approach

We now discuss the size  $n$  of TAP (the number of queries in  $Q$ ) since the size of an optimization problem usually impacts the choice of resolution approaches. Theoretically, given a cube schema, all non empty queries over this schema could be considered. Practically, it is reasonable to consider that this set is generated from a given initial query  $q_0$ .

In what follows, we restrict to star join queries over a star schema of the form  $q = (g, s, m)$  where  $g$  is the query group-by set,  $s$  is a set of selections, and  $m$  is a set of measures, all 3 sets being pairwise disjoint. Transforming a query into another with a simple OLAP operation means either changing the group-by set  $g$ , changing a selection in  $s$ , or changing a measure in  $m$ .

Even restricting  $Q$  to the set of queries that can be generated by transforming an initial query  $q_0 = (g_0, s_0, m_0)$  with a sequence of simple OLAP operations, the size of  $Q$  is potentially very large, not allowing to look for an exact solution. A rough estimate of the number of queries that can be generated from  $q_0$  by applying

$k$  OLAP operations, i.e., the size of  $Q$ , can be done by assuming for simplicity that dimensions only have linear hierarchies (no branches):

$$|Q| = ((\prod_i h_i - 1) + (|2^D| - 1) + (|2^M| - 1)) \cdot k$$

where  $h_i$  is the number of levels in dimension  $i$ ,  $D$  is the union of the active domains of all levels, and  $M$  is the set of all measures in the cube. Changing the query group-by set means picking one group-by set among all the possible ones, excluding the current one  $g_0$ . Changing the selected values means picking a set of values in the active domain, excluding the current one  $s_0$ . Changing the measure set means picking a set of measures among all possible sets of measures excluding the current one  $m_0$ .

In order to approach solutions of TAP for arbitrary large sets of queries, we adopt the following strategy. We first use a heuristic to solve the Knapsack problem and obtain a subset of queries, using estimated costs and interests, so that the estimated costs satisfy the time budget. Then, we order the queries by increasing estimated cost and evaluate them. We periodically check the difference between the time budget constraint and the elapsed time: if it is negative (too much time is taken) we redo a Knapsack to reduce the set of chosen queries; otherwise (we can benefit from additional time), we redo a Knapsack adding previously not taken queries. Finally, we determine an order on the chosen set of queries so that cognitive distance is minimized, using a heuristic to solve the TSP.

## 3 APPROXIMATION ALGORITHM

Before presenting our approach we discuss its parameters, i.e., the three functions for cost, interest, and distance, and the set  $Q$  of queries. Choosing the best three functions or defining the best set of queries  $Q$  is outside the scope of this paper. Note that a framework for learning cell interestingness in a cube is the topic of a recent paper [17]. We give examples in this section, and we indicate precisely in the tests of Section 4 the parameters used.

### 3.1 Cost

The cost of a query is related to its execution time. Classically, this cost can be estimated by a query optimizer before the execution of the query. We therefore consider that we can measure a query cost in two ways, to obtain an a priori cost (e.g., using the RDBMS optimizer) and an a posteriori cost (the actual query execution time). The a priori cost is used to decide if a query can be included or not in the solution, while the a posteriori cost is used to compute the elapsed time.

### 3.2 Interestingness measure

A crucial part of TAP lies in the definition of an interestingness measure to determine the optimal subset of queries. To quickly decide if a query is interesting, it is preferable that this measure can be computed before the actual evaluation of the query by the DBMS, therefore that it relies on the text of the query. In this sense, we propose to follow the idea of subjective interestingness measure of a pattern as developed by De Bie in the context of Exploratory Data Mining (EDM) [2] and to extend it to measure the subjective interestingness of a query as expressed by a coherent set of query parts.

In the information-theoretic formalism proposed by De Bie, interestingness is conditioned by the prior knowledge  $belief(p)$  on a pattern  $p$  of the data space, which is expressed as a probability

distribution over the set of patterns  $P$ . The interestingness measure  $IM$  is derived by normalizing the belief by the complexity of the pattern as follows:

$$IM(p) = \frac{-\log(\text{belief}(p))}{\text{complexity}(p)} \quad (1)$$

In the context of BI, [3] introduces an innovative approach to learn the belief distribution associated to query parts. The approach considers a connected graph of query parts based on the schema of the cube and the past usage of query parts, and uses a random walk on this graph to produce the expected long-term distribution over query parts. Interestingly, by construction, the probabilities obtained for each query part are independent, which allows to propose a simple formulation for the interestingness measure of a query  $q = (g, s, m)$  based on its query parts  $p \in (g \cup s \cup m)$ :

$$IM(q) = \frac{-\sum_{p \in (g \cup s \cup m)} \log(\text{belief}(p))}{|g| + |s| + |m|} \quad (2)$$

### 3.3 Cognitive distance

To order a set of queries we draw inspiration from Hullman et al. [9] who estimate the cognitive cost of transiting from the result of one query to the result of another. We use this cost as a proxy for cognitive distance. Interestingly, this cost can be estimated without having evaluated the query. Using controlled studies of peoples' preferences for different types of single visualization-to-visualization transitions, Hullman et al. [9] proposed a transition cost model that approximates the cognitive cost of moving from one visualization to the next in a sequence of static views. The transition cost is defined as the number of transformations required to convert the data shown in the first view to the second. A single transformation is defined as a change to one of the data fields shown from the first view to the second. We use this cost model to define a simple distance between queries as the Jaccard distance between sets of query parts. Formally, let  $q_1 = (g_1, s_1, m_1)$  and  $q_2 = (g_2, s_2, m_2)$  be two queries; we define:

$$\text{dist}(q_1, q_2) = 1 - \frac{|(g_1 \cup s_1 \cup m_1) \cap (g_2 \cup s_2 \cup m_2)|}{|g_1 \cup s_1 \cup m_1 \cup g_2 \cup s_2 \cup m_2|} \quad (3)$$

### 3.4 Set of queries

TAP is defined for a given set  $Q$  of queries. Practically, we consider that this set is generated from a given initial query,  $q_0$ . This initial query can be interpreted as a particular moment in an interactive analysis where the user considers what is retrieved is important and worth exploring "around" it, but has too many directions to explore. In what follows, we consider this initial query as a parameter of our approach.

Defining the set of queries worth exploring after  $q_0$  should be rooted in the earlier OLAP literature, especially automatic reporting [8], automatic cube exploration [12], discovery driven analysis [22, 23], and more generally realistic OLAP workloads [21]. In the Cinecubes approach [8], the authors consider two types of queries: (i) queries for values similar to those defining the selection filters of the initial query (i.e., siblings of ancestors), and (ii) direct drill-downs into the dimensions of the initial result, one dimension at a time. In the DICE approach [12], the authors consider direct roll-up queries, direct drill-down queries, sibling queries (a change of a dimension value, i.e., coordinate, in a dimension), and pivot queries (a change in the inspected

dimension). In the DIFF and RELAX operators [22, 23], the authors consider direct or distant drill-down (resp. roll-up) queries that detail (resp. aggregate) two particular cells of a query result. Finally, the CubeLoad OLAP workload generator [21] is based on patterns modeling realistic OLAP sessions, that could be used to generate the queries of  $Q$ .

Ideally, the set  $Q$  should include all these potentially relevant follow-up queries to  $q_0$ . For the present work we will consider different sets varying the generation of roll-up, drill-down, and sibling queries (see Section 4).

### 3.5 A simple heuristic for approximating TAP

Algorithm 1 presents our heuristic, named *Reopt*, since it is based on a naive re-optimization principle. Note that the generation of the set  $Q$  from an initial query  $q_0$  is considered as a pre-processing step. Algorithm 1 takes advantage of the fact that functions interest, cost, and distance can be computed solely on the basis of the query expression (and not on the result of the queries).

First, Algorithm 1 splits the time budget  $t$  in  $t_k$  (time for the global query execution) and  $t_o$  (time for other tasks, like solving the Knapsack, solving the TSP, etc.). Then the Knapsack is solved (line 2), and the queries selected are ordered by their estimated evaluation cost (line 3). Queries are then executed (line 6) and, after each execution, the elapsed time is checked. If it is estimated that the time budget  $t_k$  will not be respected (line 9), then another Knapsack is triggered with the remaining time. If it is estimated that the time budget will not be completely spent (line 13), then another Knapsack is triggered with all the remaining queries. Finally, once all the queries of the chosen set of queries are executed, the TSP is solved. It is easy to verify that Algorithm 1 converges: the set  $K$  is at worst  $Q$  and, at each iteration of the for loop (line 5-16), the set  $E$  is augmented with one more query of  $K$  while such query is removed from  $K$ .

Note that, when actually executing the queries, *Reopt* attaches a particular importance to the estimated cost (see for instance Line 3) compared to interest or distance. The main cause behind this decision is due to the time budget that has to be respected: had we given priority to distance from  $q_0$  or interest, we might have executed first costly queries or we might have many ties (since many queries would be at the same distance from  $q_0$ ). Although alternative formulations are also possible, due to the sensitivity of time efficiency in user interactions (keep in mind that we operate in the context of exploratory data analysis), it is imperative that the other factors do not guide query execution to take up too much time.

Implementation-wise, the Knapsack problem is solved using a Fully Polynomial Time Approximation that gives a bound on the divergence from the optimum [14], and for the TSP using the heuristic of [15].

## 4 PRELIMINARY EXPERIMENTS

Our preliminary experiments aim at understanding the behavior of our naive algorithm *Reopt*. To this end, we have compared it to two other algorithms: a brute force one that looks for the optimal solutions (named *optimal*), which obviously works only on small instances of TAP, and a simplistic one that consists of solving the Knapsack, then solving the TSP, and then executing the queries (named *K + TSP*). All the algorithms are implemented in Java 12 and run on a Core i7 with 16GB RAM under Linux Fedora. The code is available via Github (<https://github.com/OLAP3/CubeExplorer>). Note that, in all tests, costs are estimated

---

**Algorithm 1:** Reopt: simple re-optimization heuristic for TAP approximation

---

**Data:** An instance  $(Q, \text{interest}(), \text{dist}(), \text{cost}(), t)$  of TAP

**Result:** A sequence of queries

```
1 Split  $t = t_k + t_o$ 
2  $K = \text{knapsack}(Q, t_k)$ 
3 sort  $K$  by increasing cost
4  $E = \emptyset$ 
5 for each query  $q \in K$  do
6   execute  $q$ 
7    $K = K \setminus \{q\}$ 
8    $E = E \cup \{q\}$ 
9   if  $\text{elapsedTime} + \sum_{q \in K} \text{cost}(q) > t_k$  then
10    //we may have less time than expected
11     $K = \text{knapsack}(K, t_k - \text{elapsedTime})$ 
12    sort  $K$  by increasing cost
13  else if  $\text{elapsedTime} + \sum_{q \in K} \text{cost}(q) < t_k$  then
14    //we may have more time than expected
15     $K = \text{knapsack}(Q \setminus E, t_k - \text{elapsedTime})$ 
16    sort  $K$  by increasing cost
17  $S = \text{TSP}(E)$ ; // construct sequence
18 return  $S$ 
```

---

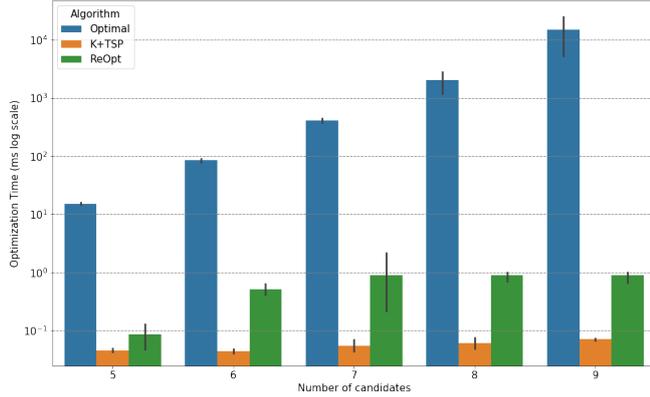


Figure 1: Optimization time

using not the estimation coming from the query optimizer of the DBMS but a linear regressive model based on the query length in ASCII, the number of tables, the number of projections, the number of selections, and the number of aggregations of the SQL star join query. This is because our empirical tests have shown that this model was more accurate than the raw estimates given by the query optimizer.

#### 4.1 Optimization time

In this first test, we use a simple synthetic cube under the schema of the SSB benchmark [20], with an instance of 1GB under MariaDB, produced with the TPC-H data generator. The cube has only one fact table and 5 dimension tables, which enables to keep the number of queries in  $Q$  under control for comparing *Reopt* and *K + TSP* with *optimal*. Over this schema, we have used the CubeLoad generator [21] to generate 40  $q_0$  queries. These queries are used to generate  $Q$  with direct roll-up or drill down queries, i.e.,  $Q$  can vary between 5 and 10.

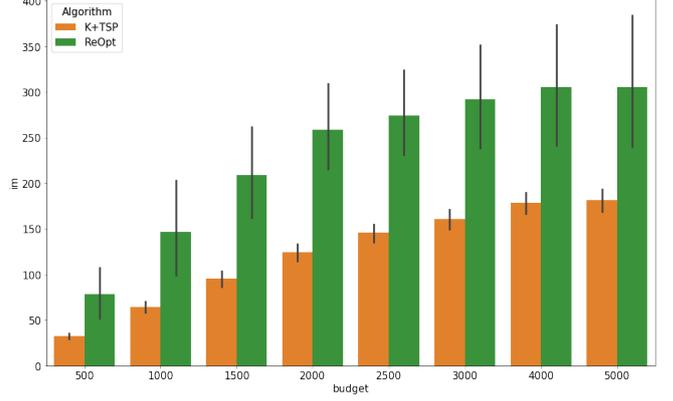


Figure 2: Total interestingness

Figure 1 shows, on a logarithmic scale, the average time taken by the 3 algorithms to optimize (not counting the query execution time) by different sizes of  $Q$  with time budget varying from 1 second to 10 seconds for *Reopt* and *K + TSP*. Results are as expected, with *K + TSP* outperforming its two competitors, since it only runs once Knapsack solving. Notably, the time taken by *Reopt* remains under control.

To assess the benefit of our re-optimization step (line 9-16 of Algorithm 1), we counted the number of times each branch of the if statement in line 9 of *Reopt* is taken, i.e., the number of times for negative (lines 10-12) and for positive (lines 14-16) re-optimizations. We have observed that both branches are used, with positive re-optimizations being rarely done compared to negative ones; precisely, over 399 total runs, positive re-optimizations are done 116 times while negative re-optimizations are done 851 times. This demonstrates that *Reopt* can adaptively respond to inaccurate estimates.

#### 4.2 Distance and interestingness vs time budget

For the next series of tests, we use a cube issued by a French project on energy vulnerability. It is organized as a star schema with 19 dimensions, 68 (non-top) levels, and 24 measures, and it contains 37,149 facts recorded in the fact table. The cube is stored in SQL Server. We use 19 real user explorations over this cube (navigation traces generated by master students when analysing data during a course) and pick as  $q_0$  the first query of the explorations. For each of these queries, we generate  $Q$  by rolling-up  $q_0$  in a dimension, drilling down  $q_0$  in a dimension, or computing a sibling query in a dimension. We use this cube and set  $Q$  to obtain more realistic instances of TAP. Precisely,  $|Q|$  ranges from 29 to 149 queries for these tests. Over these sets  $Q$  we run *Reopt* and *K + TSP* and observe how total interestingness and average distance between queries change when increasing time budget  $t$  from 500 milliseconds to 5 seconds.

Figures 2 and 3 show the results. It can be seen that *Reopt* outperforms the simplistic *K + TSP* in terms of interestingness, which illustrates the benefit of our re-optimization heuristic, while both algorithms achieve comparable average distance.

#### 4.3 Unused time

In this last test, using the same cube and protocol as in the previous subsection, we want to understand if the budget time is used properly.

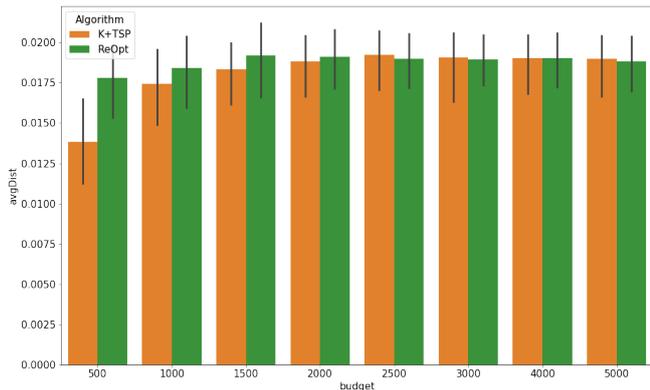


Figure 3: Average distance

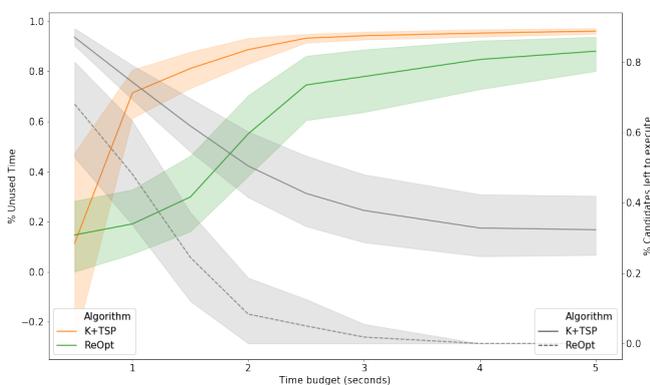


Figure 4: Unused time and candidate queries left

Figure 4 plots against different time budgets the proportion of unused time versus the number of candidate queries left in  $Q$  to execute. As we can see, regardless of the budget, our algorithm manages to take advantage of all the time available unless all queries of the  $Q$  have been explored, in which case the ratio of unused time increases. On the contrary, the  $K + TSP$  approach, which is unaware of the possible gain in executing the other queries, has a larger ratio of unused time and does not manage to explore completely  $Q$ . The absence of idle time clearly proves the advantages of our adaptive re-optimization heuristic over the static  $K + TSP$  method, which cannot compensate for the errors in the prediction of the cost, nor ensure that the execution time is near to the time budget.

## 5 CONCLUSION

This paper introduced the Traveling Analyst Problem (TAP), the problem of computing the most interesting and coherent data story that can be obtained within a reasonable time. We formalize the problem, show its strong NP-hardness and propose a heuristic for finding approximate solutions to it. Our preliminary experiments show that a heuristic based on simple re-optimization is a promising direction to obtain acceptable solutions.

We believe that TAP opens many interesting research directions. Obviously, the first step is an in-depth theoretical study of TAP, to understand which types of optimization algorithms are more appropriate. Importantly, TAP should be investigated

in the context of data exploration, which means that optimization algorithms should take advantage of classical data management optimizations, like re-optimization (e.g., [18]), and that TAP should be declaratively formulated, for instance by having starting points expressed in an intentional fashion (e.g., [24]). User tests should be conducted for further evaluating the approach. Finally, we also note that the definition of TAP is general, leaving room for variants, e.g., changing the definition of queries (e.g., from non-OLAP SQL queries to more complex actions involving queries and pattern mining or statistical tests), as well as changing cost (e.g., using self-adjusting cost model), interest (e.g., using statistics or data sampling), and distance functions.

*Acknowledgment.* The authors would like to thank Vincent T'Kindt for his insights on TAP complexity.

## REFERENCES

- [1] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem, A Computational Study*. Princeton Series in Applied Mathematics. Princeton UP, 2006.
- [2] Tijl De Bie. Subjective interestingness in exploratory data mining. In *Proc. of IDA*, pages 19–31, 2013.
- [3] Alexandre Chanson, Ben Crulis, Krista Drushku, Nicolas Labroche, and Patrick Marcel. Profiling user belief in BI exploration for measuring subjective interestingness. In *Proc. of DOLAP*, 2019.
- [4] Fabián Díaz-Núñez, Franco Quezada, and Óscar C. Vásquez. The knapsack problem with scheduled items. *Electronic Notes in Discrete Mathematics*, 69:293–300, 2018.
- [5] Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, and Naushin Shaikh. QueRIE: Collaborative database exploration. *TKDE*, 26(7):1778–1790, 2014.
- [6] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.
- [7] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proc. of NIPS*, pages 2962–2970, 2015.
- [8] Dimitrios Gkesoulis, Panos Vassiliadis, and Petros Manousis. Cinecubes: Aiding data workers gain insights from OLAP queries. *IS*, 53:60–86, 2015.
- [9] Jessica Hullman, Steven M. Drucker, Nathalie Henry Riche, Bongshin Lee, Danyel Fisher, and Eytan Adar. A deeper understanding of sequence in narrative visualization. *TVCG*, 19(12):2406–2415, 2013.
- [10] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proc. of SIGMOD*, pages 277–281, 2015.
- [11] D.S. Johnson and M. Garey. *Computers and Intractability*. W.H. Freeman, 1979.
- [12] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *Proc. of ICDE*, pages 472–483, 2014.
- [13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.
- [14] Katherine Lai and M Goemans. The knapsack problem and fully polynomial time approximation schemes (fptas). Technical report, Massachusetts Institute of Technology, 2006.
- [15] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2):498–516, 1973.
- [16] Patrick Marcel, Nicolas Labroche, and Panos Vassiliadis. Towards a benefit-based optimizer for interactive data analysis. In *Proc. of DOLAP*, 2019.
- [17] Patrick Marcel, Verónica Peralta, and Panos Vassiliadis. A framework for learning cell interestingness from cube explorations. In *Proc. of ADBIS*, pages 425–440, 2019.
- [18] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. Robust query processing through progressive optimization. In *Proc. of SIGMOD*, pages 659–670, 2004.
- [19] Tova Milo and Amit Somech. Next-step suggestions for modern interactive data analysis platforms. In *Proc. of KDD*, pages 576–585, 2018.
- [20] Patrick E. O’Neil, Elizabeth J. O’Neil, Xuedong Chen, and Stephen Revilak. The star schema benchmark and augmented fact table indexing. In *Proc. of TPCTC*, pages 237–252, Lyon, France, 2009.
- [21] Stefano Rizzi and Enrico Gallinucci. CubeLoad: A parametric generator of realistic OLAP workloads. In *Proc. of CAISE*, pages 610–624, 2014.
- [22] Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *Proc. of VLDB*, pages 42–53, 1999.
- [23] Gayatri Sathe and Sunita Sarawagi. Intelligent rollups in multidimensional OLAP data. In *Proc. of VLDB*, pages 531–540, 2001.
- [24] Panos Vassiliadis, Patrick Marcel, and Stefano Rizzi. Beyond roll-up’s and drill-down’s: An intentional analytics model to reinvent OLAP. *IS*, 85:68–91, 2019.
- [25] Abdul Wasay, Manos Athanassoulis, and Stratos Idreos. Queriosity: Automated data exploration. In *Proceedings of IEEE International Congress on Big Data*, pages 716–719, New York City, NY, 2015.