Optimisation and interactive data analysis: the Traveling Analyst Problem

Alexandre Chanson

UNIVERSITÉ DE TOURS

ÉCOLE DOCTORALE MIPTIS

Equipes BDTLN et ROOT
LIFAT (EA 6300)

THÈSE présentée par :

Alexandre CHANSON

soutenue le : **1er décembre 2023**

pour obtenir le grade de : **Docteur de l'université de Tours**

Discipline/ Spécialité : **INFORMATIQUE**

Optimisation et analyse interactive de données : le Problème du Voyageur de Données

THÈSE dirigée par :

Dr. LABROCHE Nicolas	Université de Tours
Pr. MARCEL Patrick	Université d'Orléans
Pr. T'KINDT Vincent	Université de Tours

RAPPORTEURS :

Pr. BELLATRECHE Ladjel	ISAE-ENSMA
Pr. PÉTON Olivier	IMT Atlantique

JURY :

Pr. BELLATRECHE Ladjel	ISAE-ENSMA
Pr. JOURDAN Laetitia, Présidente	Université de Lille
Dr. LABROCHE Nicolas	Université de Tours
Dr. MAABOUT Sofian	Université de Bordeaux
Pr. MARCEL Patrick	Université d'Orléans
Pr. PÉTON Olivier	IMT Atlantique
Pr. T'KINDT Vincent	Université de Tours

Acknowledgments

This thesis is part of a collaboration between the operation research (ROOT) and database (BDTLN) teams of the University of Tours computer science lab (LIFAT). I'd like to thank both teams and their members for their warm welcome. I'd like to thank the directors of both ROOT and BDTLN, Yannick and Béatrice, for their support.

To my comrades Ben, Adam, and Nicolas, who have been working alongside me on their own dissertations. Thanks for the support, the fun moments, and the jokes in the break room. Having good friends who go through the same hardship as you really helps.

To my advisors, Nicolas, Vincent, and Patrick, thanks for the many opportunities to learn and perfect the numerous skills necessary to be a young researcher. I value the time you took to help me along this journey and correct my mistakes along the way. I still have many things to learn, and hope we will have opportunities to work together in future endeavors.

To my family, your support has been essential going through this work. Three years is a long time to work on a project with ups and downs, and your support has been very much appreciated.

Résumé

Cette thèse contribue à l'automatisation de l'analyse exploratoire des données (AED). L'AED est un processus itératif qui consiste à analyser des données en effectuant des actions, telle qu'une requête sur des données, à recevoir le résultat et à décider de l'étape suivante. L'objectif final de l'AED est l'extraction de trouvailles, des fragments d'informations utiles étayées par les données. L'AED a été jusque-là un processus principalement manuel, mais depuis quelques années la communauté de recherche en bases de données a entrepris de l'automatiser. L'automatisation de l'AED nécessite de surmonter plusieurs obstacles, dont notamment l'identification et la représentation des informations les plus intéressantes présentent dans une base de données. Dans cette thèse, nous abordons le problème de l'automatisation de l'AED par la construction d'une séquence de requêtes représentant des trouvailles pertinentes. Pour se faire nous introduisons et étudions le problème d'optimisation associé, nommé le problème du voyageur de données (ou Traveling Analyst Problem, TAP). Nous établissons la relation entre le TAP et une famille de problèmes de transport classiques appelés problèmes d'orientation. Nous étudions la structures des instances du TAP et identifions des critères de domination entre requêtes que nous proposons d'exploiter. Nous proposons différentes stratégies de résolution selon les tailles des bases de données, dont notamment des matheuristiques. Celles-ci tirent profit des solveurs mathématiques pour construire des solutions de haute qualité au TAP, notamment pour les bases de données de petite taille. En étudiant la littérature, nous avons identifié l'action de comparaison de données comme la principale activité des analystes de donnée. Nous définissons formellement les trouvailles de comparaison et les requêtes de comparaison associées. Nous mettons en œuvre un prototype capable de construire les instances et de résoudre le TAP pour ces requêtes de comparaisons. A l'aide des heuristiques précédemment développées nous sommes en mesure d'appliquer notre méthode sur de grandes bases impliquant des millions de comparaisons. Pour les bases de données de plus grande taille, la simple construction des instances du TAP est une tâche complexe, nécessitant la mise en œuvre de plusieurs stratégies d'optimisation notamment pour exécuter des millions de tests statistiques rapidement. Ce processus représente la majeure partie du temps de calcul total. Afin d'éviter cela, nous nous inspirons de la méthode de génération de colonnes. Nous introduisons la génération de requêtes, une technique capable de résoudre le TAP sans explorer l'ensemble des requêtes de comparaison associées à une base de données et donc d'éviter la majorité du coût de calcul. Nous montrons sur de grandes bases de données que cette approche est non seulement plus rapide mais produit aussi de meilleures solutions que de construire la totalité de l'instance et de la résoudre à l'aide d'une heuristique traditionnelle.

Contents

List of Acronyms

Chapter 1

Introduction

Context and challenges

Exploratory Data Analysis, or EDA, is an iterative process of performing an action on a data source, such as a query on a DataBase Management System (DBMS), receiving the result, and deciding on the next step. The final goal of most EDA sessions relates to the extraction of insights. Insights can be loosely defined as fragments of useful information backed by the data. For most of its existence since the seventies, EDA has been mainly a manual process ([Tukey 1977]), and assuming a small dataset, it can even be performed on paper. However, with the increasing size and availability of open data, and businesses collecting more data than ever, this manual approach becomes quickly outdated. Data analysts are only able to focus on a few databases at a time and are prone to discovering false insights, as shown by a recent study ([Zgraggen *et al.* 2018]). For some years, the database community has been moving towards the ever-closer goal of automating EDA ([Idreos *et al.* 2015, Amer-Yahia *et al.* 2023b]). Automating EDA requires overcoming several hurdles, notably identifying and correctly presenting the most interesting insights gathered from a database.

This thesis addresses the problem of automatically constructing an EDA session as a sequence of queries representing interesting insights. Intuitively, this task can be split into four steps: (i) generating queries leading to potential insights, (ii) eliminating false insights, (iii) measuring their interestingness, and (iv) properly presenting a manageable set of insights to the user.

This problem has previously been tackled in the literature by a few pioneering works, either using specialized algorithms ([Ding *et al.* 2019]) or costly machine learning methods ([Bar El *et al.* 2019]). The former is only restricted to a specific type of insights, allowing aggressive optimizations to tackle the massive search space. The latter uses an agent exhibiting greedy behavior that largely ignores the size of the search space and lacks the elimination of false insights introduced by the former. Both elude the presentation phase relying instead on sorting ([Ding *et al.* 2019]) or generation order ([Bar El *et al.* 2019]).

We aim to provide a novel holistic approach to automated EDA. To this extent, we introduce and study the general optimization problem associated with generating EDA sessions, called the traveling analyst problem (TAP). While other works use algorithms derived from the pattern mining and machine learning community. The originality of our approach relies on the use of the knowledge and approaches of the operations research community. Including some recently introduced approaches, such as matheuristics.

The work presented in this thesis focuses on automated EDA and tackles the following questions:

- How to ensure no false insights are shown to the user?
- How to efficiently build the large search space of the TAP?
- How to model and solve the TAP to optimality? Up to which search space size?
- Can we design fast and efficient heuristics to solve TAP when optimal solutions are no longer computable?
- Can the TAP be solved without exploring the complete search space?

Contributions

Chapter 2 introduces the task of exploratory data analysis and presents the challenges and previous works relevant to its automation. Notably, we identify the comparison task as the main staple of data workers. Although our goal is to define and study a general automated EDA optimization problem, we choose this comparison task as its main application. Finally, we establish links between the generation of EDA sessions and similar problems that were tackled by the database community with the help of operations research.

Chapter 3 introduces operations research and its relevant terminology to the reader. It formally defines the Travelling Analyst Problem and establishes that the TAP is a type of orienteering problem. It discusses the relevant operation research literature. Finally, it introduces and defines the comparison query as the applicative framework of this thesis.

In Chapter 4, we propose a definition and mathematical program for the TAP and establish the relation between the TAP and a family of classic transport problems named orienteering problems. We study and generate different types of instances of the TAP. We identify dominance conditions between queries within a TAP instance. This enables us to propose cuts and a pre-filtering of TAP instances. We show that pre-filtering is beneficial to solution time while only slightly degrading solution quality. We create two heuristics based on knapsack and traveling salesperson heuristics. We test the performance of these heuristics and compare their solution quality to optimal solutions on small instances of the TAP. We elaborate on those heuristic solutions to create two additional matheuristics that deliver near-optimal solutions in a fraction of the time taken by a conventional mathematical solver.

Chapter 5, presents the main application of the TAP with comparison queries. We address several challenges linked to the generation and elimination of false insight on large databases. Heuristics introduced in Chapter 4 are successfully applied to these databases creating EDA sessions presented in interactive notebooks. We

conduct a qualitative analysis of the notebooks to identify the perceived quality of the generated EDA sessions by experts.

Finally, in Chapter 7, we propose a novel method called query generation. It is aimed at solving TAP without enumerating all possible comparison queries for a database. Indeed, for large databases, even after applying multiple optimization strategies, several hours are needed to generate the insights. This is due to the massive number of insights and costly computation involved in some interestingness measures. To address this issue, query generation takes inspiration from the process of column generation, a technique used in operations research when dealing with problem formulations with an untractable number of variables. We show that our query generation method is capable of solving the TAP without exploring the full set of comparison queries associated with a database. The query generation method is often faster and yields better results than previous solution methods for large databases.

Chapter 2

Exploratory Data Analysis

Contents

2.1 Introduction

This thesis focuses on the construction of Exploratory Data Analysis (EDA) sessions. This state-of-the-art first discusses EDA and EDA sessions. We then present optimization problems and algorithms involved in the automated construction of EDA sessions and the relevant Operation Research literature.

Exploratory Data Analysis, or the task of interactively analyzing datasets to gain knowledge, has recently attracted the interest of the database community ([Amer-Yahia *et al.* 2023b, Idreos *et al.* 2015]). However, it is not a new concept; statistician John Tukey qualified EDA as "detective's work" in his 1977 book ([Tukey 1977]). He advises the reader EDA is about using tools (i.e., statistics) to gain an understanding of the data. Modern EDA for the average data worker consists in using tools such as database query tools, specialized software, and flexible coding interfaces to gather insights ([Idreos *et al.* 2015, Amer-Yahia *et al.* 2023b]). Insights are present in most recent publications on EDA; although precise definitions may vary, they always represent a piece of useful (to a user) information about the dataset they are extracted from. This information is always somewhat aggregated, i.e., it does not concern a single tuple. Insights can be summarized in a single phrase; examples from the literature are presented in Example 1. Finally, insights can be spurious, i.e., resulting from random data and a particular aggregation ([Zgraggen *et al.* 2018]); thus several authors argue that insight should be statistically significant ([Zgraggen *et al.* 2018, Ding *et al.* 2019, Tang *et al.* 2017]). Recent work by [Ma *et al.* 2023b] goes further and links EDA to the recently popular

field of explainable Artificial Intelligence (AI) and proposes causal explanations of insights.

Example 1 (Insights from the litterature)**.**

- *rising trend of brand H's yearly increased sales [Tang et al. 2017]*
- *Factor Surgery=Yes is relevant to the difference on Lung Cancer between Location=A and Location=B [Ma et al. 2023b]*
- *For Los Angeles, April has minimum Sales [Ma et al. 2021]*
- *There are more blocked beds in the Royal London Hospital compared with the UK average [Wang et al. 2020]*

Although nowadays EDA is frequently associated with languages like Python, we insist on the importance of SQL and databases 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.

2.2 Discovery-Driven Exploration

EDA is similar to Discovery-Driven Exploration (DDE) of data cubes [Sarawagi *et al.* 1998], in whose context the pioneering works by Sunita Sarawagi ([Sarawagi 1999, Sarawagi 2000, Sathe & Sarawagi 2001]) 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 ([Sarawagi 2000]) or notable discrepancies in the data to be explained by generalization (rolling-up) ([Sathe & Sarawagi 2001]) or by detailing (drilling-down) ([Sarawagi 1999]). 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 OnLine Analytical Processing (OLAP) operations. EDA operations include data retrieval, data representation, and data mining tasks ([Milo & Somech 2018]). Finally, EDA aims to find insights into the data, i.e., high-level observations that are significant.

2.3 Interestingness of insights

Quantifying the importance of insights attracted a lot of attention in both the database and visualization communities. [Tang *et al.* 2017, Zgraggen *et al.* 2018]. It is commonly admitted that interestingness in EDA is manifold [Geng & Hamilton 2006, Marcel *et al.* 2019]. In [El *et al.* 2020], Bar El et al.

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

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 cover 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 insights extracted from datasets is often used or mixed with other interestingness measures [Tang *et al.* 2017, Zgraggen *et al.* 2018]. Tang et al. [Tang *et al.* 2017] proposed a measure combining statistical significance and the impact of the values displayed in the insights. Ding et al. with QuickInsight [Ding *et al.* 2019], uses the same approach as [Tang *et al.* 2017] for scoring insights. The impact is simply the market share of tuples used in the computation of the insight. In Metainsight [Ma *et al.* 2021], 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). DataShot [Wang *et al.* 2020] uses the same interestingness as [Tang *et al.* 2017]; the authors point to the fact that both significance and impact are easy to compute across different types of insights. With Calliope ([Shi *et al.* 2020]), Shi et al. propose to rely on information theory and compute the insight self-information and mix it with its statistical significance. Producing this measure involves computing the probability of each insight given the dataset; this is likely one of the factors preventing the authors from scaling their approach.

Overall no consensus seems to emerge from the literature over one particular interestingness measure for insights. In fact, in their recent tutorial [Amer-Yahia *et al.* 2023b] identify five separate dimensions of interestingness measures with formal definitions almost unique to each paper. Interestingness dimensions are a way to classify interestingness measures according to their goal instead of their precise mathematical definitions. They are :

1. Relevance (data vs goal): do data satisfy an information need
2. Novelty (data vs history): are data previously unknown
3. Peculiarity (data vs other data): do data differ from their peers
4. Surprise (data vs belief): data does not match user's prior beliefs
5. Presentation (data vs themselves): is data presentation intelligible

In Table 2.1, we show a selection of twelve recent publications, including our own; the interestingness measures the authors and the dimensions they pertain to. As stated by [Amer-Yahia *et al.* 2023b], there is very little agreement between authors on the dimensions an interestingness measure should pertain to. Even so, when there is an agreement, the same dimension can be represented by a different set of measures. Furthermore, there is no apparent agreement on how to combine different measures together either. Some authors opt for simple sums, while others choose products or ratios.

Reference	Interestingness	Combination	Relevance	Novelty	Peculiarity	Surprise	Presentation
[Bie 2013]	Information content, descriptional complexity	ratio				X	X
[Tang *et al.* 2017]	Significance, coverage	product			X		
[Ding *et al.* 2019]	Significance, coverage	product			X		
[Bar El *et al.* 2019]	Conciseness, distance, diversity, coherency	weighted sum		X	X		X
[Personnaz *et al.* 2021]	Familiarity, curiosity	weighted sum	X	X			
[Ma *et al.* 2021]	Conciseness, coverage	product			X		X
[Francia *et al.* 2022]	Novelty, peculiarity, surprise	weighted sum		X	X	X	
[Shi *et al.* 2020]	Significance , information content	product			X		
[Razmadze *et al.* 2022]	Coverage , diversity	weighted sum		X	X		
[Personnaz *et al.* 2022]	Uniformity, diversity, novelty	weighted sum		X	X		
[Chanson *et al.* 2022a]	Significance , conciseness, credibility	product			X		X
[Chanson *et al.* 2022b]	Learned	n/a	X				

Table 2.1: Intrestingness measures and dimensions in the recent literature

Market share Market shares comes up in several composite interestingness measures ([Ding *et al.* 2019, Tang *et al.* 2017]). Market share like interestingness itself has different definitions depending on the authors. According to [Amer-Yahia *et al.* 2023b], Market Share belongs to the coverage dimension. The simplest is the ratio of tuples that support the insights versus the total number of tuples in the database, such as in[Razmadze *et al.* 2022]. It can also be interpreted in a financial way; for example, if an insight pertains to Sales amounting to a total of $ 3 million, where the entire company sales for that year amount to $ 30 million, then the market share of this insight is 0.1 ([Ding *et al.* 2019]).

2.4 Comparisons

Several studies highlighted the importance of comparisons when analyzing data. For instance, Blount et al. [Blount *et al.* 2020] examined 67 stories produced using EDA, including award-winning data stories, from both professional journalists and data science-aware students, and found comparisons (showing multiple visualizations juxtaposed and highlighting the difference between them) to be the most popular pattern among novices and professionals alike.

Zgraggen et al. [Zgraggen *et al.* 2018] 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 that do 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 explored 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 to be able to automatically characterize comparison insights. Francia et al. [Francia *et al.* 2021] and, independently, Siddiqui et al. [Siddiqui *et al.* 2021] respectively defined the Assess and Compare operators to give a clear semantics and logical foundations of comparisons (and labeling the result of the comparison in [Francia *et al.* 2021]) 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. Chapter 5 of this thesis introduces our contribution to the automated extraction of significant comparison insights from a database.

2.5 Automating Exploratory Data Analysis

In the last few years, many attempts have been made at automating EDA [Amer-Yahia *et al.* 2023b, Milo & Somech 2020]. While the obvious time gained by the users is stated by all authors proposing automated EDA methods as a benefit. Some studies have shown that users, even experts are prone to errors when gathering insights. Rojas et al. [Rojas *et al.* 2017] studies the various sampling methods

in the context of big data exploration. They notice that users only use uniform sampling methods despite their numerous flaws. Likewise, in [Zgraggen *et al.* 2018], users are able to find "spurious" insights. Spurious insights originate in random data; by simply trying many types of aggregation, a user may come up with, for example, a mean comparison that appears to be significant. Spurious insights may even be significant with respect to a single statistical test (this is commonly known as a type I error). This is due to the Multiple Comparison Problem and is widely known in statistics and scientific fields where many statistical tests are used on the same data, such as medicine ([Lee & Lee 2018]).

Many proposals have been put forward to support data exploration, which remained until recently a mainly user-driven task ([Idreos *et al.* 2015]). They included visualization tools and interfaces, query recommendation or construction tools, and a myriad of optimization techniques to ensure the expensive analytical queries were processed in interactive time. We refer the reader to [Idreos *et al.* 2015] tutorial for an overview. Recently the database community has turned its interest toward the goal of automating EDA ([Amer-Yahia *et al.* 2023b]) to the point where any user could perform this task. This goal fits into the larger objective of automating Data Science ([De Bie *et al.* 2022]). These modern automated or semi-automated EDA approaches can be classified into two categories according to [Amer-Yahia *et al.* 2023b]. **Generate and select** approaches will extract insights, queries, or both from a dataset and pick a few to present to the user. Meanwhile, **Guided EDA** approaches are closer to the older query recommendation approaches such as [Giacometti *et al.* 2009]. They generate EDA sessions one query at a time and can often operate in a semi-automated mode.

2.5.1 Generate and select

Tang et al. ([Tang *et al.* 2017]) proposed an approach for extracting trend insights from multidimensional data. They compute the top-k insights, i.e., having the highest interestingness measure. Unlike pattern mining, their measure is non-monotonic, excluding traditional algorithms such as Apriori ([Agrawal & Srikant 2000]). They instead exploit the multidimensional structure of the data to order and reuse computation as well as introduce pruning rules. Ding et al. proposed QuickInsight [Ding *et al.* 2019], a system for discovering a broad spectrum of insights (Changepoint, Correlation, Outlier, Seasonality, etc.) in multidimensional data. Unlike [Tang *et al.* 2017] QuickInsight does not rely on an underlying multidimensional database; it instead uses SQL and can interact with any DBMS. Quickinsight also relies on ordering and caching to improve performance. An implementation of QuickInsight has been integrated into the production version of Microsoft Power BI[2]. Metainsight [Ma *et al.* 2021] aims at organizing the knowledge conveyed by insights extracted from multidimensional data in terms of commonness and exception. They present their insight in an ordered fashion like [Tang *et al.* 2017], but instead of relying solely on interestingness, they also consider the inter-pattern similarity and

[2] https://learn.microsoft.com/en-us/power-bi/create-reports/service-insights

potential overlaps. DataShot [Wang *et al.* 2020] aims at extracting insight and displaying the most informative in an infographic sheet. The authors conducted an extensive survey of expertly crafted infographics to associate various insights types (trend, outlier, etc) with visualization types. They limit their search for insights involving at most three attributes based on empirical observations. Unlike other authors, Young et al. do not discuss the performance of their algorithms. This is likely due to the small datasets used by the authors (at most 30k tuples and 9 attributes) and the hard limit on the search space. The Travelling Analyst Problem (TAP) belongs to the **generate and select** family. Indeed the TAP consists of an optimization problem defined by an exhaustive set of queries or insights needed to be arranged in a sequence of maximum interest, minimum time, and minimum overall distance.

2.5.2 Guided EDA

Guided EDA method set themselves apart as their behavior is closer to older query recommendation tools and human analysts than **Generate and select** methods. Indeed they approach the EDA problem iteratively. Constructing sessions one query at a Time. The first two approaches in this category, [Bar El *et al.* 2019] and [Personnaz *et al.* 2021], rely on reinforcement learning (RL). They define a set of operations either set oriented ([Personnaz *et al.* 2021]) or using the multidimensional nature of the data ([Bar El *et al.* 2019]). These operations, combined with a starting point, allow the agent to construct an EDA session. The agent acts to maximize a reward function. In both works, it consists of a multi-faceted interestingness measure that takes into account previous queries, which encourages coherent sessions. The main drawback of those approaches is the considerable training time, especially when it is repeated for each dataset. Another issue is the lack of ability to discard a previous query if a better candidate is found later in the exploration. This greedy one-query-at-a-time behavior can also be an advantage as it is easy to use these methods as an assistant for semi-automated EDA where the user might only want to generate one or two queries during its own session. The third approach, Calliope ([Shi *et al.* 2020]), improves on DataShot by proposing a more efficient algorithm. Calliope proposes a logic-oriented Monte Carlo tree search algorithm that explores the data space and progressively generates insights, and organizes them in a logical order. Like RL-based approaches, at each iteration, it maximizes a reward function. It builds a tree where nodes are insights; and edges are transitions. A path from the root to a leaf represents a sequence of insights the algorithm may output. At each iteration, the tree can be altered in four ways: Select finds a node with the largest reward, Expansion search for insights that are logically relevant, Simulation explores several steps further, searching for the direction with the largest reward, Back-propagation updates the weights of the nodes. Like DataShot, the output of the algorithm is organized visually in an infographic sheet. Despite the advantages of not having to enumerate all insights and having zero training time, the authors only seem to demonstrate their algorithm on very small datasets (at most 1200

tuples and 6 attributes).

With the growing prevalence of Large Language Models (Large Language Model (LLM)) in the AI community ([Zhao *et al.* 2023]), we may soon witness the emergence of EDA tools [Amer-Yahia *et al.* 2023a] based on LLMs. Such as [Ma *et al.* 2023a], which augments EDA sessions produced by a traditional approach ([Tang *et al.* 2017]) with summaries created using an LLM. In this pioneering work, the authors leverage Microsoft's previous works on **generate and select** insight extraction ([Ding *et al.* 2019, Ma *et al.* 2021]). The LLM serves as an interface between their traditional insight extraction algorithms and the user. It runs the insight extraction in four different modes according to the perceived user intent and provides a text summary of the discovered insight(s). Although this first work is promising, some issues, such as data privacy and hallucination, may need to be resolved before LLMs become widespread in EDA tasks [Amer-Yahia *et al.* 2023a]. In the future, we might witness automated EDA approaches that are solely based on LLM. Some data analysts have already taken it upon themselves to use LLM for that purpose ([Sawtell-Rickson 2023]).

2.5.3 EDA as an optimization problem

While **guided EDA** approaches can be compared to single query recommendation, **generate and select** approaches are close to another well-studied recommendation problem, composite item recommendation. Composite Item (CI) address complex information needs and are prevalent in problems where items should be bundled to be recommended together [Amer-Yahia & Roy 2018], like in task assignments in crowdsourcing or travel itinerary recommendation. CI formation is usually expressed as a constrained optimization problem, and different CI shapes require the specification of different constraints and optimizations. Our formulation of the TAP appears to be related to chain-shaped CIs (e.g., touristic itinerary, etc.) [Roy *et al.* 2011, Cao *et al.* 2012, Gionis *et al.* 2014], that are traditionally 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 points of interest popularities), this last one often being used as the objective function. Retrieval of chain CIs is usually NP-hard, being reduced to TSP or orienteering problems, and has been addressed through greedy algorithms [Roy *et al.* 2011, Cao *et al.* 2012], dynamic programming or dedicated TSP strategies [Gionis *et al.* 2014].

Our guided EDA problem appears close to chain CI recommendation. However, these works often include a specific starting and ending point (e.g., hotel). We, nonetheless, follow the same strategy as authors working on CIs. In the next chapter, we formally define the TAP as an optimization problem, then we relate it to the current operations research literature on transport problems with profits. Finally, we discuss possible solution methods based on our analysis of the literature.

Chapter 3

The Traveling Analyst Problem

Contents

3.1 Introduction

The TAP was first described in [Chanson *et al.* 2020]. While some automated EDA approaches ([Amer-Yahia *et al.* 2023b]) may generate queries as they construct a session, the TAP involves constructing an ordered session from a known finite set of queries Q. This set can be composed of any number and class of queries. In this thesis, we consider Q to be the set of comparison queries (see Section 2.4) over a given database.

Given this set of queries Q, the goal is to construct a sequence of queries. We assume each query has an interest and a cost to run, and the distance between queries is known. Thus, the goal is to produce a session of maximum interest, minimum overall distance, and minimum cost. This can be formalized using the following graph representation.

Definition 1 (TAP). *Let $G = \langle Q, E \rangle$, with $E = Q \times Q$, be the complete, directed, labeled graph where each node corresponds to a query in Q. For each vertex $q_i \in Q$, there are two labels: p_i, its interest value, and t_i, the cost associated with running the query. Likewise, for each arc (q_i, q_j), there is a label d_{ij} corresponding to the distance between the two queries q_i and q_j[1]. The objective of the TAP is to produce a sequence $s = [q_a, q_b ... q_{final}]$ over G, with no repetitions, such that the overall interestingness*

[1] In this work, we assume distances are symmetric but may not always respect the triangular inequality

score $\bar{P}(s) = \sum_{i=1}^{|s|} p_i$ *is maximum, the overall distance* $\bar{D}(s) = \sum_{i=1}^{|s|-1} d_{i,i+1}$ *is minimum and the cost* $\bar{T}(s) = \sum_{i=1}^{|s|} t_i$ *is minimum.*

As mentioned in Section 2.5.3, some problems close to the **guided EDA** can be related to the traveling salesperson problem. The traveling salesperson problem and many of its generalizations have been extensively studied by the operation research community. By looking into its definition, we can classify the TAP as a TSP with profits. Specifically, since its interest and cost objectives cannot be combined and the interest cannot be expressed as a constraint, it belongs to the family of the Orienteering Problem (OP) ([Feillet *et al.* 2005]). Like other extensions of the TSP, the OP allows for a selection of nodes to be visited with a specific reward associated with visiting a given node. The length of the circuit is often associated with a constraint (e.g., the maximum range of a vehicle).

This chapter is organized as follows. We first discuss operations research and the general principle for solving optimization problems. We then introduce literature pertaining to the OP. Next, we introduce matheuristics, a fairly recent type of heuristics. Then, we introduce the comparison queries that are used throughout this thesis. Finally, we present a summary of the contributions of this thesis.

3.2 Operation research in a nutshell

Operations research (OR) is one of the oldest fields in computer science, emerging in the 20th century and growing during the Second World War. During this time, OR was used by allied military planners to better allocate assets and conduct more efficient combat operations ([Kirby 2003]). Modern OR deals with the development of analytical methods to solve decision or optimization problems. In this introduction, we focus on optimization problems and solution methods. Throughout this section, we use the well-known knapsack problem to introduce the methods and the terminology used by the OR community.

Example 2 (0-1 Knapsack Problem)**.**
Given a set O *of* n *objects, represented by their values* $p_j \in \mathbb{N}\ \forall j \in O$ *and weights* $w_j \in \mathbb{N}\ \forall j \in O$*, and* $W \in \mathbb{N}$ *the capacity of the knapsack.*
We search for $O^* = \arg\max_{O' \in O} \sum_{k \in O'} p_k$ *such that :* $\sum_{k \in O'} w_k \leq W$

Instance and solution The set of all inputs used to solve an optimization problem is referred to as an instance. For example, $I_1 = \langle O = \{Ham, Banana, Milk\}, W = 12, p_{Ham} = 4, p_{Banana} = 6, p_{Milk} = 10, w_{Ham} = 8, w_{Banana} = 8, w_{Milk} = 4\rangle$ is an instance of the Knapsack Problem. For a given instance, solutions to the problem can then be found. Here, given I_1, the optimal solution O^* to this instance is $\{Banana, Milk\}$ with an objective value of 16. Unless specific proof is given, there is no guarantee of the uniqueness of the optimal solution to an optimization problem. Any solution that satisfies the problem's constraints is

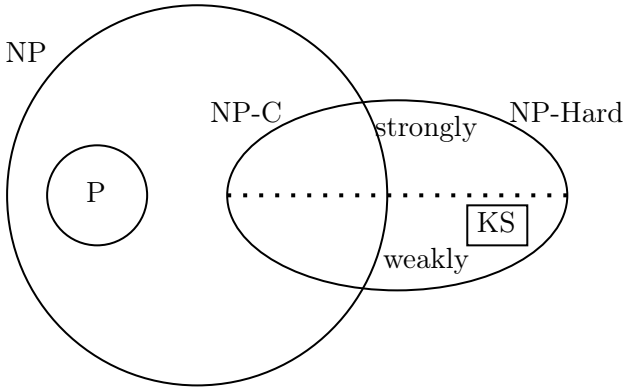

Figure 3.1: Common complexity classes.

referred to as a feasible solution (e.g., $\{Ham, Milk\}$).[2]

Complexity Complexity theory provides us with a way of quantifying the time taken by a computer to solve a problem. This time complexity can be given as a function of the size of the problem instance. For our knapsack problem, the size of the instance n is the number of objects, i.e., $n = |O|$. Another quantity of interest given a problem instance is m, the magnitude of the largest number in the instance, i.e., $m = W + \sum_{k \in O} p_k + w_k$ for the knapsack problem. Simple problems such as finding the minimum of a convex function fall into the $\mathcal{P}$ class (Figure 3.1): they are problems optimally solvable in polynomial time of n. While $\mathcal{NP}$-Hard contains hard problems such as the knapsack problem or the traveling salesperson problem ([Garey & Johnson 1990]), they cannot be solved in polynomial time of n. The knapsack problem, in particular, has been shown to be *weakly* $\mathcal{NP}$-Hard as there are algorithms ([Kellerer *et al.* 2004]) which solve it in polynomial time of n and m: such algorithms are referred to as pseudo-polynomial time algorithms. On the other hand, *strongly* $\mathcal{NP}$-Hard problems (Figure 3.1) cannot even be solved by pseudo-polynomial algorithms.

When dealing with hard optimization problems, solution strategies are divided into two broad categories: exact methods and heuristics.

3.2.1 Exact solution methods

There are many categories of exact methods. Dynamic programming-based algorithms can be designed for problems that can be broken down recursively into sub-problems. Dynamic programming algorithms take advantage of this structure to efficiently reuse computations. For the knapsack problem, we can recursively solve sub-problems involving the first j items of O. A dedicated algorithm that does not rely on dynamic programming can also be designed to solve a specific problem.

[2] in this manuscript, we use the term instance to refer to the instance of an optimization problem while, slightly abusing instance of a relation and the instance of a problem, whenever possible, we refer to term relation for relation instance

Knapsack has many dedicated algorithms, such as the expanding core algorithm ([Pisinger 1995]).

Mathematical programming is a universal tool to model optimization problems and can be used to construct a first exact solution algorithm. Let us construct a mathematical model for our 0-1 Knapsack problem. First, we need a set of binary variables to represent the choice of selecting an object or not:

$$\forall k \in O, x_k = \begin{cases} 1 & \text{if object } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

Then, the objective function, we want to maximize the profit selected items bring:

$$\sum_{k \in O} x_k p_k \tag{3.1}$$

Finally, we impose to our formulation to the capacity constraint of the knapsack :

$$\sum_{k \in O} x_k w_k \leq W \tag{3.2}$$

This particular mathematical model is referred to as a linear binary Integer Program (IP), as it only contains binary variables and linear objectives and constraints. There are many types of mathematical models like Mixed Integer linear Programs (MIP), which contain a mix of integer and real variables, or Linear Programs (LP), which only involve real variables. There are several more types of models we choose to omit, including models based on quadratic objectives, constraints, or both; we only focus on LP and MIP as they are used in this thesis.

Note that there exists an important relationship between MIP and LP. Relaxing the integrality requirements on all the integer variables of a MIP (e.g., allowing binary and integer variables to take real values in their domain) creates an LP we refer to as the LP relaxation of the MIP. Solving to optimality, this LP relaxation yields an upper bound in the case of a maximization problem on the objective value of the optimal solution to the original MIP. Likewise, for a minimization problem, the LP relaxation yields a lower bound.

Solution of LP and MIP Solutions to an instance, when represented using a mathematical model, consists of the values of all its variables. Thus, the optimal solution for I_1 is $(w_{Ham} = 0, w_{Banana} = 1, w_{Milk} = 1)$. Linear programs can be solved in polynomial time ([Schrijver 1986, 172-180]). In practice, the simplex algorithm and its many variants are commonly used by commercial ([IBM 2021]) and open-source solvers[3], despite its exponential worst-case time complexity. For solving generic IP and MIP, no polynomial algorithms can exist unless $\mathcal{P} = \mathcal{NP}$

[3] https://www.gnu.org/software/glpk/

([Schrijver 1986, 245-248]). However, generic exponential algorithms can successfully solve MIP with up to thousands of variables and constraints. Commercial solvers, such as CPLEX ([IBM 2021]) and Gurobi ([Gurobi 2016]), and open-source implementations, such as CBC[4], use variations of a branch-and-cut algorithm to solve any problem with some integer variables.

Branch-and-cut methods are themselves a variation of the branch-and-bound algorithm. Both rely on the progressive construction and exploration of a search tree. Leaves represent solutions where all variables are integers (e.g., for the knapsack, we have decided to pick or not to pick each object) but not all leaves represent feasible solutions. At the root node, an LP-Relaxation gives a bound on the objective function and real values for each variable (e.g., for the knapsack, no object has been picked or left out yet). Intermediate nodes are partially integral solutions where some variables have been set to an integer (e.g., for the knapsack, some objects are selected or not).

Solving the first LP relaxation at the root node, it is unlikely that all variables have integer values, and we get the optimal solution without any work. Most of the time, we have to start construction/exploration of the tree. From there, we keep track of the best-known MIP feasible solution and use it as our lower bound. First, choose a non-integral variable (e.g., with a value of 0.5) and branch creating two nodes, one with the variable a value of 1 and the other with the variable having a value of 0. Add those two nodes to the search tree. While the search tree is not empty :

1. Pick a node on the search tree

2. Create an LP relaxation with the remaining free variables and solve it

3. Use the optimal LP solution to try to prune the node: (a) the LP is infeasible, prune the node. (b) the optimal LP solution value of the node is lower than the current lower bound, prune the node. (c) the optimal LP solution of the node is feasible to the MIP (e.g., all variables have integer values). Update the current lower bound and the best-known MIP solution, and prune the node.

4. If we cannot prune the node. We must branch. Choose a non-integral variable and branch, creating two nodes. Add those two nodes to the search tree.

This is an oversimplified version of the algorithm used by modern solvers when dealing with MIP. The Branch-and-cut variant adds special constraints to the LP-relaxation named cuts that lower the gap between the optimal solution values of the MIP and its relaxation. All steps of this algorithm have to take into account many considerations that can dramatically change its performance. One of them is the order in which to consider the nodes in the search tree. Improvements to branch-and-cut methods to solve specific or generic MIP have been the topic of thousands of publications in the OR community.

[4] https://coin-or.github.io/Cbc/intro

3.2.2 Heuristics

Many $\mathcal{NP}-$Hard problems require solutions for instances far beyond the size that can be solved in a reasonable amount of time by exact methods. In this case, heuristic methods are used: they cannot guarantee to find an optimal solution but rather feasible solutions. Heuristics are usually designed for a specific problem, following a more general principle such as Tabu Search or greedy heuristics.

Constructive heuristics iteratively build a solution. Greedy algorithms fall into this category. A simple constructive heuristic for the knapsack problem can be constructed by sorting objects in decreasing order according to p_k/w_k and inserting them in order until W is reached.

Branching heuristics are similar to exact branch-and-bond methods in their use of search trees to represent partial solutions. However, they only explore a fraction of the search tree, losing all guarantees offered by their exact counterparts. For example, the beam search only allows exploring b nodes on a level discarding others based on a given criterion.

Finally, neighborhood-based heuristics rely on iteratively improving solutions. They are often paired with a constructive or branching heuristic to construct a "starting" solution. Alternatively, this initial solution may be randomly constructed. At each iteration, neighborhood-based heuristics explore a neighborhood. It consists of a limited set of solutions derived from their current solution(s). Simply varying the definition of what constitutes the neighborhood of a solution, or introducing diversification techniques, yields many variations on this principle. Well-known heuristics such as tabu search, simulated annealing, and genetic algorithms belong to this category.

Depending on the type of problem to be solved and the time and memory resources available to solve it, one or more heuristic strategies may be designed. Some heuristics may perform well on specific instances of a problem while yielding poor performance on others.

Overall, heuristics represent an incredibly diverse and useful set of tools for solving optimization problems. Recently the OR community has extended this set of tools by adding a new type of heuristic that makes use of mathematical programming and mathematical solvers: Matheuristics.

Matheuristics Among the heuristics available in the operations research community, matheuristics have been a matter of growing interest in the last decade [T'Kindt 2023]. Matheuristics are a type of heuristics that make use of mixed integer programming. [Fischetti & Fischetti 2018] define matheuristic as "the hybridization of mathematical programming with metaheuristics". Matheuristics can be categorized into three classes ([T'Kindt 2023]):

- constructive matheuristics iteratively build a solution like constructive heuristics by using mathematical programming;
- local search matheuristics use a MIP to improve a known solution;

- evolutionary matheuristics embed the solution of a MIP into an evolutionary algorithm.

Local search matheuristic, like their traditional heuristic counterparts, are defined by their way of constructing a neighborhood. They can be divided into two broad categories depending on their definition of neighborhood. On one hand, local branching matheuristics rely on the distance to a known solution to define a neighborhood [Fischetti & Fischetti 2018]. Typically, this is done through the variables of a MIP. For example, with the 0-1 knapsack problem, we can use the values of the binary variables of two solutions and compare them using Hamming distance ([Hamming 1950]). This distance is then used in the MIP to create a constraint limiting the feasible solution to the neighborhood. On the other hand, matheuristic such as Variable Partitioning Local Search (VPLS) or Fix and Optimize rely on freezing a set of variables in a known solution, leaving the solver to modify only parts of it ([Della Croce *et al.* 2013]). The selection of those variables depends on the problem. The VPLS scheme has proved to be efficient on hard permutation problems ([Della Croce *et al.* 2013, T'Kindt 2023]). All local search matheuristics include intensification phases where the iteratively produces a higher-quality solution through local search. However, Local Branching also includes a diversification step that can be triggered when no improving solution has been found in the neighborhood at the previous iteration.

3.3 The Orienteering problem

The OP was first introduced and described by [Tsiligirides 1984]. The name is derived from the orienteering sport, which involves competitors running from a starting point to a finishing point within a specified time frame. Along the way, they must pass through control points, each of which carries a different reward. Formally, the OP is defined on a graph where each node is given a reward value and each edge a distance. The objective is then to find a path (or tour) with maximum collected reward between a known start (and end) node. A bound is set on the total distance traveled between nodes (or total travel time) along this path.

Over the years, many extensions of the OP have been put forward. Recent surveys [Vansteenwegen *et al.* 2011, Gunawan *et al.* 2016] have identified a few main extensions of the orienteering problem. The simplest extension, called the Team Orienteering Problem (TOP) involves multiple agents collaborating to gather rewards, with the added constraint that the reward from a node can only be gathered once. Adding time windows when the reward is available on nodes yields another variant, called the Orienteering Problem with Time Windows (OPTW). The OPTW can also be combined with multiple agents and is named Teams Orienteering Problem with Time Windows (TOPTW).

For the OP, [Tsiligirides 1984] proposed two heuristics to solve it. The first heuristic is based on a monte-carlo process generating several feasible solutions and choosing the best one. The second heuristic is based on an earlier vehicle routing

heuristic by [Wren & Holliday 1972]. It relies on circular subdivisions of the Euclidean space to construct solutions. A local search procedure called route improvement ([Tsiligirides 1984]) is then used to improve solutions from both algorithms. Route improvement alternatively tries to introduce new vertices without exceeding the distance budget or shortening the path length by changing the order of vertices in the route. [Chao *et al.* 1996] propose a two-step heuristic that first builds a set of feasible solutions exploiting geometric features before applying a local search procedure to improve them. This procedure consists in exchanging vertices between the current best solution and other feasible solutions to improve the former. Both authors use geometric assumptions about the instances, namely a classic Euclidean geometry. While Chao et al. also assume that if a straight path is drawn between start and end vertices, then the high-value vertices should be further away from this path than the low-value ones. This makes a direct adaptation of the heuristics for the TAP difficult, although we note that both authors successfully use local search to improve existing solutions, which may be useful in our work.

Several works describe exact methods for solving the OP. In [Fischetti *et al.* 1998], a branch-and-cut algorithm is proposed and tested on many instances, including those from [Tsiligirides 1984] but also larger ones with up to 500 vertices. They manage to solve those large instances to optimality in several hours. Most of the extensions of the OP tackled in the literature turn out to be more complex to solve to optimality. [Bianchessi *et al.* 2018], propose a branch-and-cut algorithm to solve the Team Orienteering Problem (TOP). This algorithm relies on a MIP formulation, with a polynomial number of variables and constraints, along with a custom branch-and-bound procedure. The authors are able to optimally solve instances up to 102 vertices and 4 vehicles. Two MIP formulations for the OP are proposed in [Kara *et al.* 2016] with a polynomial number of constraints and variables. Both formulations are used to solve the instances proposed by [Chao *et al.* 1996] and [Tsiligirides 1984]. However, they are not tested on the larger instances of [Fischetti *et al.* 1998]. [Kara *et al.* 2016] differ from previous works by using no custom branching routine, being solely reliant on the solver to interpret their MIP formulation.

[Hu & Lim 2014], Hue and Linn tackle an extension of the OP, called the Team Orienteering Problem with Time Windows (TOPTW), in which a set of identical vehicles (with the same travel capabilities) are considered. A vertex can be visited by only one vehicle, and the vertices can only be visited within their specified time windows. Hu and Lim propose a metaheuristic to solve this problem. Several routes are generated, one for each agent. Those routes are then stored in a fixed-size pool. Routes from this pool are combined to form complete solutions for the problem. The algorithm also features several operators enabling the crossover of routes or swaps of vertices in a route to improve the built solutions. This work, along with [Chao *et al.* 1996] and [Tsiligirides 1984], points out the importance of reordering vertices when constructing solutions heuristically.

Matheuristics Matheuristics have been used to solve extensions to the OP. [Archetti *et al.* 2015],tackle the arc routing team orienteering problem by using a matheuristic, which combines a tabu search and a MIP solver. The algorithm is able to solve 78% of the tested instances to optimality. In [Yu *et al.* 2019], Yu et al. focus on time-dependent profits and provide a matheuristic that yields high-quality solutions for instances with up to 200 vertices. This algorithm first solves the problem of sequencing vertices before using a MILP solver to find appropriate service times for the computed sequence.

Relation to the TAP Overall, the TAP is closer to the classic OP than its variants. However, the TAP differs from the OP in a few key aspects. First, by adding a knapsack constraint linked to a kind of service time necessary to collect rewards on nodes. Secondly, although the OP often involves paths instead of tours depending on the application for the TAP, every node is a potential end or start point. This formulation is quite close to that of [Roy *et al.* 2011], but the latter simplifies the problem by merging service times and travel times. In the case of EDA, the distance has semantics in itself and cannot be made analogous to a travel time or to a physical distance. Thus, it must be considered separately. These differences appear to come from the fact that most applications of the OP and its extension are to actual physical transport problems and not a conceptual one like TAP.

3.4 Comparison queries

As the first contribution of this thesis, we introduce the comparison query. We choose to focus our work on this operation as it is a major component of EDA sessions ([Blount *et al.* 2020]). Comparisons are extremely popular among data workers ([Zgraggen *et al.* 2018, Blount *et al.* 2020, Siddiqui *et al.* 2021]). Furthermore, by limiting our work to a pattern of queries, we ensure the search space for the TAP is finite and we can compute its cardinality.

Since comparisons frequently happen in practice, with a high risk of comparison-based insights being spurious ([Zgraggen *et al.* 2018]), there is a need to automate the production of non-spurious comparison insights represented by comparison queries.

Our work complements previous works in EDA ([Tang *et al.* 2017, Ding *et al.* 2019]) that address other forms of insights. Noticeably, while we restrict here to comparison insights, our approach can be extended to other forms of insights, such as correlations.

Given a relation of schema $R[A_1, ...A_{n_a}, M_1, ...M_{n_m}]$with $\{A_1, ...A_{n_a}\}$ the set of categorical attributes, noted $\mathcal{A}$, likewise $\mathcal{M}$ for the set of measures $\{M_1, ...M_{n_m}\}$. We note $dom(A_i)$ the active domain of attribute $A_i \in \mathcal{A}$. We also consider $\mathcal{F}$ the set of aggregation functions. With a slight abuse of notations, R denotes both the relation name and an instance of the relation, while $|R|$ denotes the cardinality of

```
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 3.2: A SQL comparison query, its result, and an insight

the instance R.

We assume attribute names and values are two disjoint sets of constants with total orders. Furthermore, the function $D : \mathcal{A} \rightarrow \mathbb{N}$ returns the number of constants in the active domain of a given categorical attribute.

Definition 2 (Comparison query)**.** *The comparison query considered in this work is phrased in extended relational algebra ([Garcia-Molina et al. 2002, 189-237]) as follows:*
$\tau_A((\gamma_{A,agg(\alpha)\rightarrow left}(\sigma_{B=val}(R))) \bowtie (\gamma_{A,agg(\alpha)\rightarrow right}(\sigma_{B=val'}(R))))$
where A is a categorical attributes taken in $\mathcal{A}$, α is a measure in $\mathcal{M}$, $agg \in \mathcal{F}$ is an aggregate function, and B another categorical (distinct from A). Finally, val and val' are values in the domain of B.

For simplicity purposes, 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.

Note that our definition of comparison queries requires that a tabular presentation is used for the presentation of the result, hence the need for the join and the projection in the outermost position. Alternatively, comparison queries could be written without joins: $\gamma_{A,B,agg(M)}(\sigma_{B=val \vee B=val'}(R))$. However, this would require a pivot operation ([Cunningham *et al.* 2004]) to present the result in a suitable tabular way. [Francia *et al.* 2021] experimented with the two forms, which appeared to be similar in terms of execution cost.

Lemma 1 (Number of comparison queries)**.** *For a relation R where f aggregation functions can be applied. We note Q, the set of all possible comparison queries over R that follows the pattern of Definition 2. Its cardinality is:*

$$\sum_{i=1}^{|\mathcal{A}|} \binom{|dom(A_i)|}{2} \times (|\mathcal{A}|-1) \times |\mathcal{M}| \times f$$

Example 3 (Sequence of comparisons)**.**
An example of a comparison query for the analysis of COVID-19 infections is given in Figure 3.2. When applying TAP to comparison queries, its solution is a sequence of comparison queries. It could start with the query of Figure 3.2; the next query would change the measure (replacing cases by deaths). The next one changes the aggregation function (replacing sum by average), then changes the selection by comparing months 5 and 6. Finally, the last query could group by countries instead of continents (keeping the same selections).

3.5 Summary of contributions

The contributions of this thesis are organized into two categories and three chapters. First, we discuss cases where we generate the complete set Q of all comparison queries on a database. Then we propose to forgo this step and solve the TAP without generating Q.

3.5.1 Enumeration of the complete set of queries

Our first contributions, presented in Chapters 4 and 5, are focused on building and solving instances of TAP with hundreds to hundreds of thousands of queries. We propose a MIP model and use a solver to solve a diverse panel of artificial and real TAP instances with up to 500 queries. We propose a matheuristic approach to speed up this process and allow instances with up to 700 queries to be solved with close to optimal solutions in minutes. We then propose two metaheuristics capable of scaling to instances with hundreds of thousands of queries. Finally, we apply TAP to a real-world EDA scenario where even the construction of the problem is shown to be unpractical for the largest databases.

3.5.2 Intractability of the enumeration of the whole set of queries

Our last contribution, described in Chapter 6, aims to propose an alternative to the lengthy generation of the TAP instances while maintaining the quality of solutions offered by a complete enumeration of the search space. We also wish to provide an alternative to costly reinforcement learning based approaches and greedy algorithms (see Chapter 2). To this extent, we draw inspiration from the column generation approach used in the OR community. We propose a method to build an instance orders of magnitude smaller than the complete instance and solve it using the matheuristics

previously introduced. We show that our proposed method is faster and produces better solutions than solving an entire instance of the TAP heuristically.

Chapter 4

Results on enumerable space

Contents

4.1 Introduction

In this chapter, we formally define the TAP problem and propose optimal solutions through solving a Mixed Integer Programming (MIP). Additionally, we introduce heuristics inspired by Knapsack and Travelling Salesperson Problem (TSP) heuristics and matheuristics for the TAP.

Assuming that we dispose of a set of database queries Q, the TAP can be defined on a graph $G = \langle V, E \rangle$ in which each vertex $v_i \in V$ represents a query. An edge $(v_i, v_j) \in E$ represent the action of running query v_i before v_j. To meet the three requirements of EDA sessions, we introduce for each vertex $v_i \in V$ an interestingness score p_i and a service time t_i. Besides, for each edge $(v_i, v_j) \in E$, we introduce d_{ij} as the distance between the two queries. The objective of the TAP is to produce a routing s over G such that the overall interestingness score $\bar{P}(s) = \sum_{i=1}^{|s|} p_i$ is maximum, the overall distance $\bar{D}(s) = \sum_{i=1}^{|s|-1} d_{i,i+1}$ is minimum and the service time $\bar{T}(s) = \sum_{i=1}^{|s|} t_i$ is minimum. When there is no ambiguity regarding the routing/solution s we use $\bar{P}, \bar{D}$ and $\bar{T}$ instead of $\bar{P}(s)$, $\bar{D}(s)$, and $\bar{T}(s)$. The TAP has been shown to be strongly NP-Hard in [Chanson *et al.* 2020].

The TAP is a multi-objective optimization problem. Thus the optimal solution to a TAP instance is not unique but a set of incomparable solutions called Pareto optima ([T'kindt & Billaut 2006]).

Definition 3 (Pareto Set of TAP)**.** *Let $\mathcal{S}$ be the set of feasible solutions to a TAP instance. The set of Pareto optima of a TAP instance is $\mathcal{P} = \{s \in \mathcal{S} : \nexists s' \in \mathcal{S}, \bar{P}(s') \geq \bar{P}(s) \wedge \bar{D}(s) \geq \bar{D}(s') \wedge \bar{T}(s) \geq \bar{T}(s')$, with at least one strict inequality$\}$. A solution $s \in \mathcal{P}$ is called a Pareto optimum.*

In this work, we assume that the user can formulate and adjust a bound on $\bar{T}(s)$ based on the time they allow for query execution. Likewise, they can use a bound on $\bar{D}(s)$ to create more or less explorative sessions (depending on the distance function chosen) that either explore a few dimensions and measures or allow a broader picture of the database. This enables us to use the ε-constraint method ([T'kindt & Billaut 2006]) to compute a Pareto optimum for the TAP: we maximize $\bar{P}$ under the constraints that $\bar{T} \leq \varepsilon_t$ and $\bar{D} \leq \varepsilon_d$. This method enables the enumeration of all Pareto optima by solving this constrained problem with different $(\varepsilon_t, \varepsilon_d)$ values.

This chapter first will introduce a MIP model for the TAP problem, then in Section 4.3 discuss methods for reducing instance size. In Section 4.4, we introduce heuristics and matheuristics adapted to TAP. Finally, Section 4.5 provides detailed experiments evaluating the heuristics using artificial instances.

4.2 A mixed integer formulation

Let us introduce a MIP formulation of the $\varepsilon-$constrained TAP. This model relies on two sets of binary variables to represent vertices selection and sequencing of the selected vertices. We have:

$$\forall i \in 1..n, y_i = \begin{cases} 1 & \text{if vertex } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\forall i, j \in 0..n+1, i \neq j, x_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ precedes vertex } j \\ 0 & \text{otherwise} \end{cases}$$

We also introduce integer variables $u_i \in \{2, ..., n\}$, $\forall i \in 1..n$, used for sub-tour elimination.

Objective

$$max \sum_{i=1}^{n} p_i y_i \tag{4.1}$$

Constraints

$$\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} d_{i,j} x_{i,j} \leq \varepsilon_d \tag{4.2}$$

$$\sum_{i=1}^{n} t_i y_i \leq \varepsilon_t \tag{4.3}$$

$$\sum_{i=0, j\neq i}^{n} (x_{i,j}) - y_j = 0, \forall j \in 1..n \tag{4.4}$$

$$\sum_{j=1, j\neq i}^{n+1} (x_{i,j}) - y_i = 0, \forall i \in 1..n \tag{4.5}$$

$$\sum_{j=1}^{n} x_{0j} = \sum_{i=1}^{n} x_{i,n+1} = 1 \tag{4.6}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), \forall i, j \in 1..n, i \neq j \tag{4.7}$$

This model involves $(n^2 + 5n + 1)$ variables and $(n^2 + 2n + 4)$ constraints. The objective (4.1) aims to maximize the total score, i.e. the interestingness of the sequence of vertices. Constraint (4.2) ensures that the total distance does not exceed a threshold ε_d. Similarly constraint (4.3) ensures that the total service time does not exceed a threshold ε_t. Constraints (4.4) and (4.5) ensure the solution is a path (if a vertex is selected in the solution, then one arc must enter it and one must exit it). Constraint (4.6) ensures there is only one start and one end vertex. Finally, we use classic TSP sub-tour elimination constraints (4.7) to ensure a single sequence is computed. Here, we chose those presented in [Miller *et al.* 1960].

4.3 Preprocessing the set of queries

The multi-objective nature of the TAP is useful as it enables us to propose a method for reducing the sizes of instances before solving them. In order to do so we first need to establish several properties of the TAP instances, notably a dominance condition on vertices.

Definition 4 (Dominance condition)**.** $\forall v_i \neq v_j \in V, v_i$ *dominates* v_j ***iff*** $t_i \geq t_j, p_j \leq p_i$ *and* $\forall v_k \in V \setminus \{v_i, v_j\}, d_{ik} \leq d_{jk}$*, with at least one strict inequality.*

Lemma 2 (Dominated vertices in Pareto optima)**.** $\forall s \in \mathcal{P}, \forall v_i \in s, \forall v_j \neq v_i \in V$*,* ***if*** v_j *dominates* v_i ***then*** $v_j \in s$

Proof: Let $s \in \mathcal{P}$ *be a Pareto optimum and two vertices* v_i *and* v_j *such that* $v_i \in s$*,* $v_j \notin s$ *and* v_j *dominates* v_i*. Construct* s' *by replacing* v_i *by* v_j *in* s*. Then, we have* $\bar{P}(s') \geq \bar{P}(s), \bar{D}(s') \leq \bar{D}(s)$ *and* $\bar{T}(s') \leq \bar{T}(s)$ *with at least one strict inequality, which contradicts the fact that* $s \in \mathcal{P}$*.*□

This dominance can be exploited when solving the TAP, as it yields constraints that simplify the problem, if selecting a vertex in the solution all vertices dominating it must also be included. However, as this dominance condition is unlikely to be satisfied in most instances due to the very restrictive condition on distances, we propose another condition.

Definition 5 (Pseudo-dominance condition)**.**

$$\forall v_i \neq v_j \in V, v_j \textit{ pseudo-dominates } v_i \textit{ } \textbf{\textit{iff}} \textit{ } t_j \geq t_i, p_j \leq p_i \textit{(Also noted } v_i \prec v_j\textit{)}$$

With at least one strict inequality.

Unlike the dominance condition stated in Definition 4, the pseudo-dominance stated in Definition 5 is not a dominance condition as applying it may lead to discard optimal solutions. Applying the pseudo-dominance condition we define the notion of pseudo dominance set.

Definition 6 (Pseudo-dominance set)**.** *Given an instance, let the pseudo-dominance set of a vertex v_j be:*

$$I_j = \{v_i \in V : v_j \neq v_i, v_j \prec v_i\}$$

As previously mentioned, the dominance condition is unlikely to happen in practice. The pseudo-dominance condition may be used to prune the search space in the context of a heuristic solution of the TAP. The benefits of using pseudo-dominance conditions will be evaluated in Section 4.5. Pseudo-dominance conditions can be used during the solving process as follows: whenever a vertex v_j is selected, all vertices in its pseudo-dominance set I_j are also selected.

Since this work focuses on solving the ε-constrained TAP, we can exploit those ε-constraints (equations (4.2), (4.3)) along with the pseudo-dominance conditions in order to filter any given instance.

Let N be an upper bound on optimal solution sizes, i.e., such that $|V| \geq N \geq |s|$ for any optimal solution s. In this chapter, N is computed as $N = min(k, k')$, where k and k' are two bounds calculated by exploiting the ε-constraints of the problem. Assume that service times are ordered such that $t_1 \leq ... \leq t_{|V|}$. Then, k is defined as $\sum_{j=1}^{k} t_j \leq \varepsilon_t < \sum_{j=1}^{|V|} t_j$. Besides, k' is obtained by solving a relaxation of the TAP in which all p_i are set to 1, the $\varepsilon-$constraint on $\overline{T}$ is dropped along with sub-tour elimination constraints. An optimal solution to this relaxed problem is found in polynomial time by constructing the shortest 2-cycles between vertices until ε_d is reached.

For any given instance we can use the pseudo-dominance sets of vertices and the bound N to design a filtering step: remove vertices v_i such that $|I_i| > N$, as any vertex with a pseudo-dominance set larger than N is unlikely to be in an optimal solution. Remember that this filtering is not optimal as the pseudo-dominance condition may lead to considering vertices as dominated while they are part of some optimal solutions. However, we will see in practice how impacting is this filtering.

Pseudo-dominance condition based constraints Using Definition 5, we also propose to add an additional set of constraints to the MIP defined in Section 4.2.

$$y_i \leq y_j, \forall i \in 1..n, \forall j \in I_i \tag{4.8}$$

However, it is important to mention that adding those constraints may lead to the optimal solution being infeasible as they are based on the pseudo-dominance condition. We evaluate the impact of adding these constraints in Section 4.5.

4.4 Heuristics

In this section, we present two heuristics and four matheuristics to solve the TAP. We introduce the heuristics first, as matheuristics rely on their solutions to start. The importance of those initial solutions given to the matheuristics is crucial, as the matheuristics can be seen as metaheuristics, which iteratively improve a base solution. Hopefully, the latter is obtained quickly and is sufficiently good so that the matheuristic can reach near-optimal solutions in a few iterations. Furthermore, when the matheuristic cannot be used, the initial solutions may be the only available solutions to a TAP instance.

4.4.1 Initial heuristics

We first introduce two different heuristics to construct TAP solutions. The first one, called *h-ks*, runs in $O(n^2)$ time and exploits the knapsack structure of the problem. Algorithm 1 introduces nodes (queries) in the TAP solution according to the decreasing order of their ratio p_i/t_i, until constraints (4.2) and (4.3) prevent for more insertions. In addition to this classic knapsack heuristic, whenever a vertex is selected, it is inserted at the position in the partial solution, which minimizes the $\bar{D}$ criterion of the selected solution under construction.

The second algorithm, called *h-tsp*, is an approach inspired by sub-tour merging heuristics for the traveling salesperson problem (TSP). This algorithm produces two solutions which are compared, and the best one is returned. The algorithm (see Algorithm 2 and Figure 4.1) starts by solving a relaxed version of the TSP over G with distances d_{ij} when sub-tour elimination constraints are removed. This yields an assignment problem that is solved by the successive shortest path method ([Engquist 1982]) in $O(n^3 log(n))$ time (Line 1). This may lead to a potentially non-feasible solution composed of sub-tours (see step ⓐ Figure 4.1). From this point, the algorithm computes two solutions. First to compute, **Solution A**, the algorithm merges all sub-tours (see step ⓑ Figure 4.1). This step is skipped if the assignment problem solution comprises a single tour (Line 4). Sub-tours are merged by implementing the minimum spanning tree approach described in [Kahng & Reda 2004] (Line 6). This merging approach was chosen as it outperforms other tour construction heuristics ([Kahng & Reda 2004]). Since after merging sub-tours, the distance and/or time ε-constraint may be violated, we design an additional

Algorithm 1 Knapsack inspired Heuristic (h-ks)

Require: A set of nodes Q with their interest t_i, time q_i and distances d_{ij}, and two reals ε_t (time bound), ε_d (distance bound).

Ensure: a feasible solution to the TAP, of total service time at most ε_t and overall distance at most ε_d

for $q \in Q$ **do**
  $weight(q) \leftarrow p_q/t_q$
end for
$\mathfrak{Q} \leftarrow$ sort Q by weights in decreasing order
$t \leftarrow 0$
$S \leftarrow []$
for $q \in \mathfrak{Q}$ **do**
  $\mathfrak{S} \leftarrow$ the set of possible unique insertions of q in S
  $d_{min} \leftarrow min_{S' \in \mathfrak{S}}(\sum_{i=1}^{|S'|} d_{i,i+1})$
  if $t + t_q < \varepsilon_t$ **and** $d_{min} < \varepsilon_d$ **then**
    $S \leftarrow argmin_{S' \in \mathfrak{S}'}(\sum_{i=1}^{|S|} d_{i,i+1})$
    $t \leftarrow t + t_q$
  end if
end for
return S

Algorithm 2 TSP inspired Heuristic (h-tsp)

Require: A set of nodes Q with their interest t_i, time q_i and distances d_{ij}, and two reals ε_t (time bound), ε_d (distance bound).

Ensure: a feasible solution to the TAP, of total service time at most ε_t and overall distance at most ε_d

$T \leftarrow solve\_assignment(Q, dist)$ ▷ We get a set of sub-tours by solving the assignment problem for the given vertices and distance function
—**Solution A**—
if $|T| = 1$ **then**
  $S_a \leftarrow T$
else
  $S_a \leftarrow$ merge(T)
end if
if $\sum_{i=1}^{|S_a|} d_{i,i+1} > \varepsilon_d$ **or** $\sum_{i=1}^{|S_a|} t_i > \varepsilon_t$ **then**
  $S_a \leftarrow prune(S_a)$
end if
—**Solution B**—
$T' \leftarrow$ solve multi-dimensional knapsack with sub-tour as elements
$S_b \leftarrow merge(T')$
return $best(S_a, S_b)$

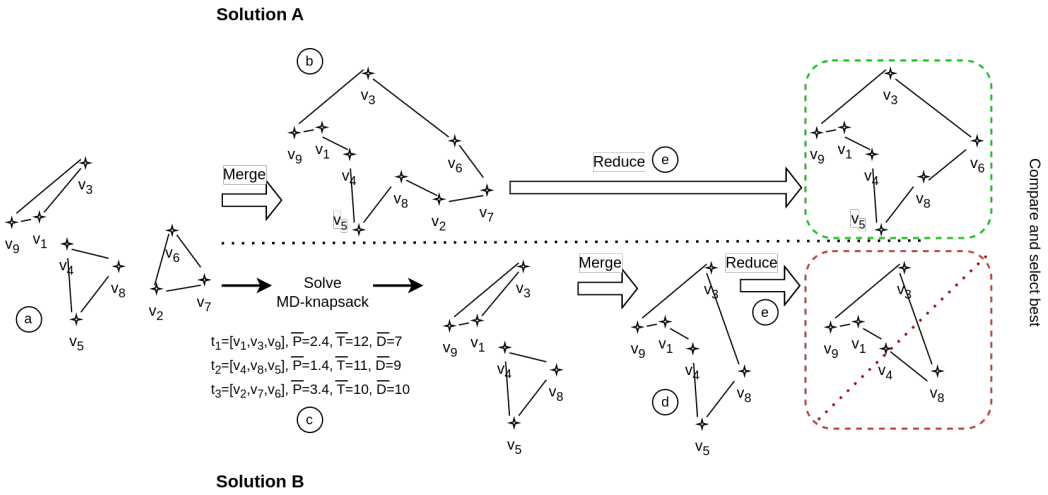

Figure 4.1: General principle of h-tsp for generating Solution A (top) and Solution B (bottom) starting from a solution to the assignment problem over the distance matrix

reduction step to fix this issue (see step ⓔ Figure 4.1, Line 9). First, if the ε-bound on time is not answered, queries are eliminated in increasing order of their ratio p_i/t_i until it is no longer violated. Then, if the ε-bound on distance is not respected, queries are eliminated in increasing order of p_i value, every η eliminated queries the LKH heuristic [1] reoptimizes the tour ([Helsgaun 2000]).

Solution B is obtained by considering all sub-tours as single elements in a Multi-Dimensional Knapsack. Each sub-tour is associated to its total length, service time, and interestingness score (see step ⓒ Figure 4.1). This Multi-Dimensional Knapsack problem is solved to extract a set of sub-tours that are next merged (see step ⓓ Figure 4.1, Line 12) using the same merging approach as Solution A (Line 13). The same reduction step as **Solution A** can be applied if any ε-constraint is violated.

Finally, **Solution A** and **Solution B** are compared, and the best (feasible) one in terms of $\bar{P}$ value is returned.

4.4.2 Matheuristics

In this section, we propose four matheuristics to solve TAP. They belong to two distinct families of matheuristics. The first two are based on the VPLS principle, and the other two use a local branching constraint. We briefly discuss both approaches and present the four algorithms.

[1] We use the implementation provided by Helsgaun `http://webhotel4.ruc.dk/~keld/research/LKH/`

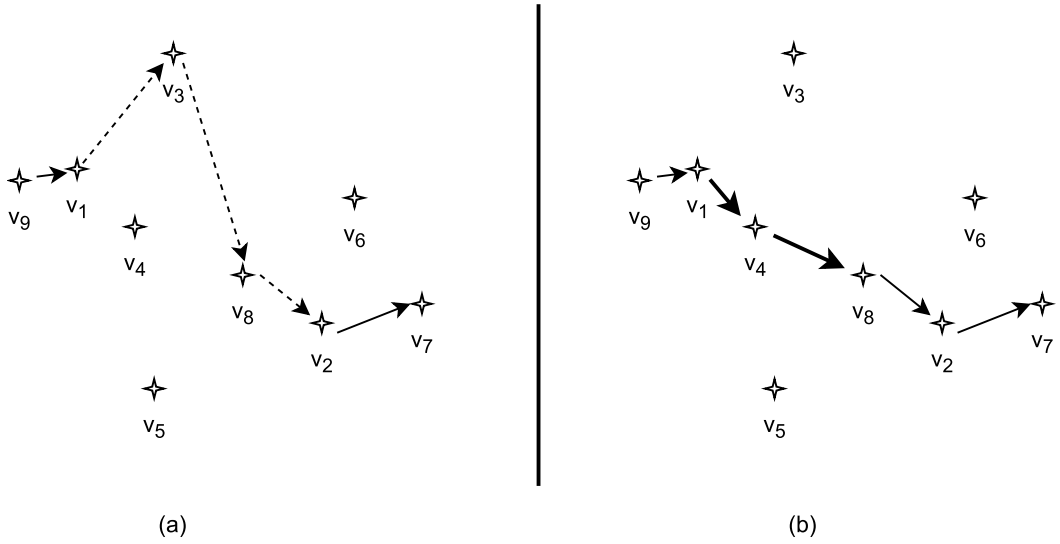

Figure 4.2: Example of a solution improved by the VPLS algorithm

4.4.2.1 VPLS

The VPLS method improves an incumbent solution by iteratively reoptimizing a part of it. As the re-optimization is done via the solution of a MIP, at each iteration, two sets of variables are defined. One set of variables is fixed as in the incumbent solution, while the variables of the other set are set free so that they define a small MIP to be solved. Hopefully, this new small MIP can be solved quickly. In the case of the TAP, the solution is a sequence: thus, we apply the VPLS method by selecting a sub-sequence of vertices (called the re-optimization window) and by freezing the remaining vertices of the solution. Given a solution s^t at iteration t, let us denote by w_{start} (respectively w) the starting position (respectively length) of the window. Given a TAP solution s^t, let $s^t[a : b]$ be the sub-sequence starting from position a and ending at position b (inclusive). The set of variables fixed to their current values for iteration $t + 1$ is defined as:

$$\begin{aligned} F_{t+1} = \{&y_i, i \in s^t \setminus s^t[w_{start} : w_{start} + w]\} \\ \cup \{&x_{ij}, i \in s^t \setminus s^t[w_{start} : w_{start} + w], j \in 1..n, j \notin s^t[w_{start} : w_{start} + w]\} \\ \cup \{&x_{ji}, i \in s^t \setminus s^t[w_{start} : w_{start} + w], j \in 1..n, j \notin s^t[w_{start} : w_{start} + w]\} \end{aligned} \tag{4.9}$$

VPLS is an iterative process that can be stopped by a specific convergence criterion like a maximum time limit or a maximum number of iterations.

We provide an example in Figure 4.2. Figure 4.2.a shows the current feasible solution $s^t = [v_9, v_1, v_3, v_8, v_2, v_7]$ at iteration t, together with the unselected vertices $\{v_4, v_5, v_6\}$. Assume that we decide to reoptimize the window $[v_3, v_8]$, which is represented by dotted arrows. Figure 4.2.b shows $s^{t+1} = [v_9, v_1, v_4, v_8, v_2, v_7]$ which is the solution obtained after solving the corresponding MIP with a reduced set of variables. One of the key points of VPLS is the choice of the re-optimization

window at each iteration. We propose two methods, and both assume that the window length w is a given parameter. The first heuristic, called *vpls-det*, described in Algorithm 3, moves the re-optimization window from the start to the end of the sequence. An additional parameter o may induce an overlap of windows between iterations if set to a non-zero value by the user. This window is positioned back at the beginning of the sequence when, at an iteration t, the solution is improved. The second heuristic called *vpls-random* and described in Algorithm 4, randomly selects the window along the sequence.

Algorithm 3 vpls-det

Require: An instance (scores, service times, distances, ε_t, ε_d), the window width w, window overlap o, m_{it} the maximum number of iterations, and t_{it} the maximum iteration time. A feasible solution s^f

Ensure: A solution to the TAP of service-time at most ε_t and overall distance at most ε_d.

$t \leftarrow 0$
$s^t \leftarrow s^f$
$w_{start} \leftarrow 0$
while $t < m_{it}$ **do**
    $s^{t+1} \leftarrow MIP(s^t, w_{start}, w, t_{it})$ ▷ solve reduced MIP
    if $\bar{P}(s^{t+1}) > \bar{P}(s^t)$ **then**
        $w_{start} \leftarrow 0$
        $s^t \leftarrow s^{t+1}$
    else
        $w_{start} \leftarrow w_{start} + w - o$
    end if
    if $w_{start} + w > |s^t|$ and $\bar{P}(s^t) = \bar{P}(s^{t+1})$ **then**
        return s^t
    end if
    $t \leftarrow t + 1$
end while
return s^t

4.4.2.2 Local branching

The two other matheuristics we introduce are direct applications of the Local Branching ([Della Croce *et al.* 2013]) method with no diversification phase. They rely on the Hamming distance ([Hamming 1950]) between the two sets of decision variables of the MIP, as described in Section 4.2: variables y_i represent the presence or absence of a vertex in the solution while the $x_{i,j}$'s represent the order between selected vertices. Thus, any two solutions can be compared by their Hamming distance, either on the complete set of decision variables or on a subset. For example, assume it is computed w.r.t. only the x_{ij}'s, then the Hamming distance $\Delta(x, x^s)$ to a known solution x^s is given by:

Algorithm 4 vpls-random

Require: A TAP instance (scores, service times, distances, ε_t, ε_d), the window width w, m_{it} the maximum number of iterations, and t_{it} the maximum iteration time. A feasible solution s^f

Ensure: A solution to the TAP of service-time at most ε_t and overall distance at most ε_d.

1: $t \leftarrow 0$
2: $s^t \leftarrow s^f$
3: **while** $t < m_{it}$ **do**
4: $w_{start} \leftarrow$ pick a random window starting position
5: $s^t \leftarrow MIP(s^t, w_{start}, w, t_{it})$ ▷ solve reduced MIP. Note that the solution returned is at least as good as the original in terms of objective.
6: $t \leftarrow t + 1$
7: **end while**
8: return s^t

$$\Delta(x, x^s) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \oplus x^s_{ij} \tag{4.10}$$

With $\oplus$ the binary XOR operator, or without (a formulation more adapted to a MIP solver) :

$$\Delta(x, x^s) = \sum_{i,j=1, x^s_{ij}=1}^{n} (1 - x_{ij}) + \sum_{i,j=1, x^s_{ij}=0}^{n} x_{ij} \tag{4.11}$$

This distance can be used to build a constraint that effectively constrains the solver to search in the neighborhood of an incumbent solution. In contrast to the VPLS approaches, which are limited to modifications within a specific sub-sequence, this approach enables broader transformations of the solutions, such as swapping the first vertex and the last vertex of the solution. The two local branching heuristics we propose are denoted by *lb-y* and *lb-yx* and are described in Algorithm 5. They simply vary on the constraints they use, (4.12) for *lb-y* and (4.13) for *lb-yx*. Let s^t be the best solution known at iteration t and let (x^t, y^t) be its associated binary variables in vector form. Then, we can express the following constraints:

$$\Delta(y, y^t) < h \tag{4.12}$$

$$\Delta(x, x^t) + \Delta(y, y^t) < h \tag{4.13}$$

with h a parameter limiting the maximum number of variables to be changed.

Algorithm 5 lb-y / lb-yx

Require: A TAP instance (scores, service times, distances, ε_t, ε_d), the maximum hamming distance h, m_{it} the maximum number of iterations, and t_{it} the maximum iteration time. A feasible solution s^f. A hamming distance based constraint: $\mathcal{C}$, either constraint (4.12) or (4.13).

Ensure: A solution to the TAP of service-time at most ε_t and overall distance at most ε_d.

1: $t \leftarrow 0$
2: $s^t \leftarrow s^f$
3: **while** $t < m_{it}$ **do**
4: $\quad s^t \leftarrow MIP(s^t, \mathcal{C}, t_{it})$
5: $\quad t \leftarrow t + 1$
6: **end while**
7: return s^t

4.5 Experiments

We organize this experimental section as follows: first, we describe different types of TAP instances and their properties; second, we evaluate the performance of the MIP in a state-of-the-art solver; third, we focus on the impact of filtering and adding constraints. Finally, we focus on the performance of heuristics and matheuristics.

For all experiments, we used CPLEX solver version 20.10 running on a Fedora Linux workstation, with two 2.3 Hz Intel Xeon 5118 with 377GB of memory. To show realistic running times, the CPLEX solver is run in single-thread mode whenever a MIP model has to be solved. We provide an open-source implementation of all algorithms in our Git repository `https://github.com/AlexChanson/TAP-Matheuristics`. This repository also contains all TAP instances used in this chapter.

To the best of our knowledge, no suitable benchmark for our problem is available in the OR literature. This is due to the added service time and the use of a distance with no triangular inequality. Although some databases are commonly used in works pertaining to EDA since there is no agreement on a 'universal' interestingness measure or even distance between queries, we cannot find a suitable benchmark in the database literature either. Instead, we use four families of randomly generated instances.

- The first family of instances, denoted by *f1*, is designed to mainly ressemble knapsack instances. The interestingness score is a real number drawn from a uniform distribution between 0 and 1. The distance is an integer number drawn from a uniform distribution between 5 and 6. Finally, the service time is an integer number drawn from a uniform distribution between 5 and 50.

- The second family of instances, denoted by *f2*, is designed to mainly ressemble TSP instances. The interestingness score is an integer number drawn from

a uniform distribution between 1 and 3. The distance is an integer number drawn from a uniform distribution between 1 and 14. Finally, the service time is an integer number drawn from a uniform distribution between 5 and 6.

- The third family of instances, denoted by *f3*, is designed to contain particularly hard knapsack instances (service time and interest are strongly correlated [Pisinger 2005]), while still having a strong routing component. The distance is an integer number drawn from a uniform distribution between 1 and 10. The service time is an integer number drawn from a uniform distribution between 5 and 50. The interestingness score is equal to the service time plus 5. Note that, due to this correlation, the pseudo-dominance conditions are never satisfied for any pair of queries in those instances.

- The last family of instances, denoted by *f4*, are closer to real world instances. The interestingness score is a real number drawn from a uniform distribution between 0 and 1. The distance is an integer number drawn from a uniform distribution between 1 and 10. Finally, the service time is an integer number drawn from a uniform distribution between 5 and 50. These constants are coherent with the ranges observed in real-world instances we generated in one of our papers ([Chanson *et al.* 2022a]).

For each family of instances, we generate 30 instances for each size $n \in \{40, 60, 80, 100, 200, 300, 400, 500, 600, 700\}$.

4.5.1 Computation of optimal solutions

As previously stated, we focus on developing algorithms solving ε-constrained TAP instances. In this series of experiments, we intend to fix ε_t and ε_d to simplify the interpretation of results. However, we must select values that do not yield TAP instances that are trivial to solve. Due to the varying size of instances, we express ε_t and ε_d as the expected fraction of queries in the instances in the solution. For example, if ε_t is set to 0.35 for 200 vertices, it means we expect that the time constraint will, on average, lead to solutions containing 70 queries. By choosing different values for ε_t and ε_d we can make one $\varepsilon-$constraint more restrictive than the other.

We speculated that some combinations of $\varepsilon-$constraints may result in instances that are relatively trivial to solve. We measured the time taken by CPLEX solving the MIP proposed in Section 4.2 to produce an optimal solution on 30 instances of *f4* with 300 nodes, with various $\varepsilon-$constraints combinations. We report those results in Figure 4.3. We decide on ε_t set to 0.6 and ε_d set to 0.3 for the remainder of this work. As this appears to be a hard scenario for the solver, but it is still practically possible to obtain optimal solutions in one hour or less.

Finally, we repeat the previous experiment and increase the instance size to 500 nodes. As seen in Figure 4.4, with most configurations solving the MIP takes on average an hour making practical scaling beyond 500 node instances impossible.

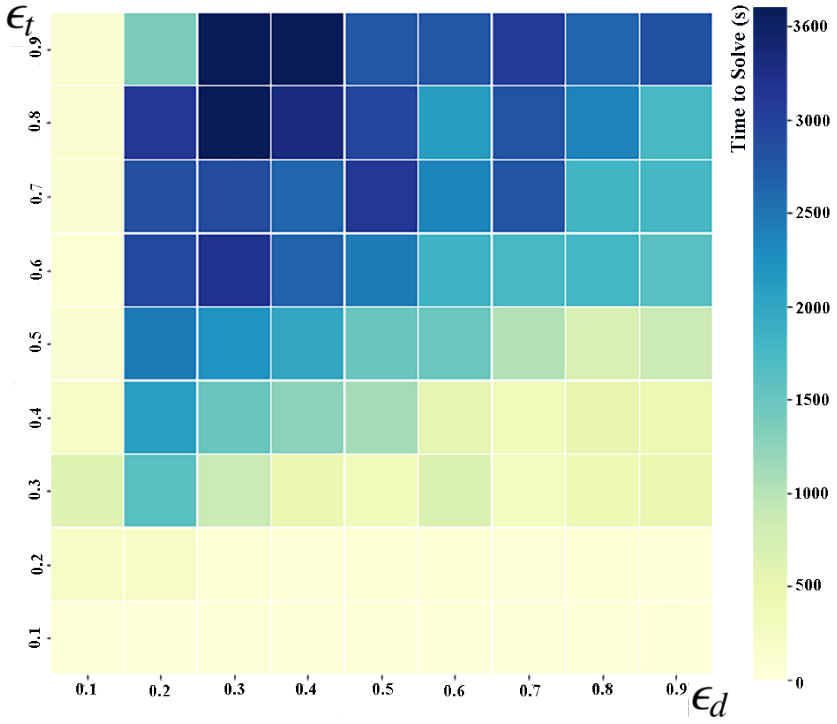

Figure 4.3: Average time taken by CPLEX to solve an instance (300 vertices) from set *f4* to optimallity

Table 4.1: Time taken by CPLEX to solve the TAP to optimality on different instance sizes

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

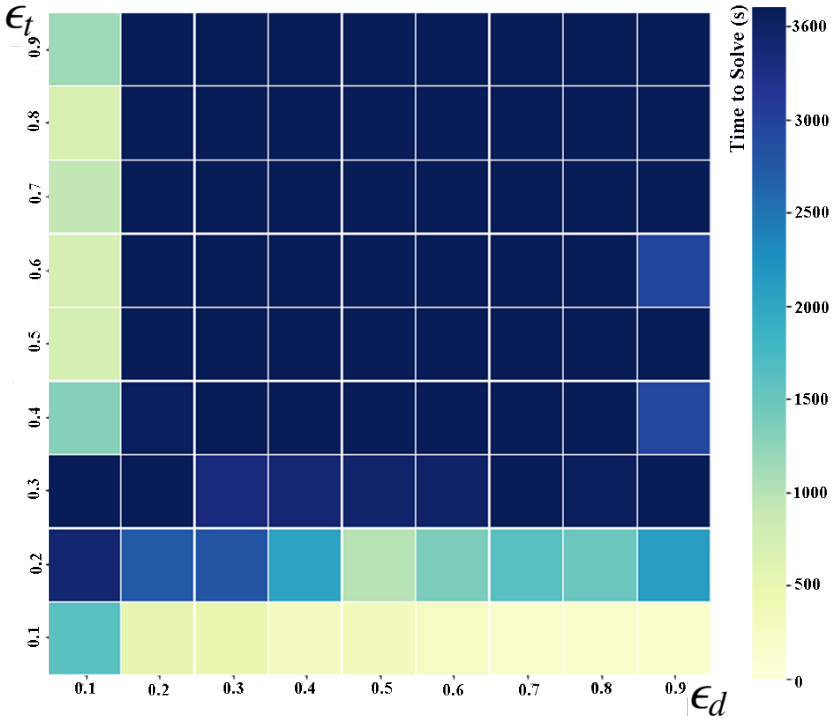

Figure 4.4: Average time taken by CPLEX to solve an instance (500 vertices) from set *f4* to optimallity

For a fixed pair of $\varepsilon-$constraint ($\varepsilon_d = 0.4$, $\varepsilon_t = 0.2$), we scale the instances up to 700 nodes (30 instances per size). We report the runtimes (minimum, average and maximum) and number of timeouts in Table 4.1.

4.5.2 Evaluation of pseudo-dominance and filtering

In this section, we evaluate the effectiveness of filtering and constraints based on pseudo-dominance conditions. Both approaches were proposed in Section 4.3. We only conduct experiments on instances of the family *f4*, as for those of family *f3* pseudo-dominance conditions never apply.

First, we propose to evaluate the impact of adding constraints based on the pseudo-dominance condition to the solution of the MIP. We use CPLEX with default parameters and a timeout of two hours, with and without these conditions. In Table 4.2 we report the average and maximum number of nodes #Nodes explored by CPLEX, and the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value (computed without pseudo-dominance). Finally, we report the average and maximum time gained ΔT_{avg} and ΔT_{max} compared to CPLEX without pseudo-dominance: negative values denote longer run-time over the vanilla solver. In a set of preliminary experiments, it was noted that the pseudo-dominance conditions produced a large number of constraints, potentially slowing the solver. Thus, we only report results where we add a fraction of the valid constraints, which

is achieved by only selecting constraints that relate to vertices with dominance sets larger than $\kappa \times |V|$, with $\kappa \in \{0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

Table 4.2 shows that adding the constraints (even partially) seems to marginally degrade the solution quality. However, this approach always requires more CPU time than CPLEX to solve instances. Even with very few added constraints. We note, though, that the best-case scenario seems to be for $\kappa = 0.8$.

As we may eventually use pseudo-dominance conditions in the matheuristics, we test the solution quality of CPLEX alone, against CPLEX with added constraints (for $\kappa = 0.8$) in a limited 60 seconds time budget. Typically, this could be the time taken by one iteration of the matheuristics. Solution quality is reported in terms of average relative deviation to the optimal solution in Table 4.3.

Size	Δ_{avg} CPLEX ($<$ 60s)	Δ_{avg} CPLEX ($<$ 60s) with pseudo-dominance ($\kappa = 0.8$)
40	0	0
60	0	0
80	0	0
100	0.05	0.14
200	30.23	9.31
300	91.11	90.71
400*	99.69	99.68

Table 4.3: Average relative deviation to the optimal solution objective after 60 seconds for various instance sizes (* deviation reported against near-optimal solutions, MIP gap $< 1\%$)

Results from Table 4.3 show that the approach can provide some significant improvement to solution quality for instances up to 200 nodes. While they seem to have no effect on larger instances. As the matheuristics where these constraints may be applied will eventually be used to tackle larger instances of TAP (>300 nodes) where the mathematical solver is too slow, we choose to discard this approach for the remainder of this work.

For the filtering approach (see Section 4.3), a preliminary test was first conducted to evaluate the number of vertices the filtering step removes on instances of family *f4*. The results are reported in Table 4.4. This shows that applying the method based on the computed bound N only filters out about 3% of the vertices. This is unlikely to be enough to achieve significant performance improvements.

κ	Size	Δ_{avg}	Δ_{max}	ΔT_{avg} (s)	ΔT_{max} (s)	#Nodes$_{avg}$	#Nodes$_{max}$
0	40	0	0	0	0	0	0
	60	0	0	0	0	0	0
	80	0	0	0	0	0	0
	100	0	0	0	0	0	0
	200	0	0	0	0	0	0
	300	0	0	0	0	0	0
0.2	40	0	0	-0.74	0.2	69	580
	60	0	0	-1.51	0.84	129	1443
	80	0	0.01	-1.86	17.21	118	1191
	100	0	0	-45.02	51.98	575	3629
	200	0	0.01	-1003.11	1743.38	1702	5950
	300	1.98	21.83	-2363.51	404.39	1720	4265
0.3	40	0	0	-0.39	0.44	66	630
	60	0	0	-1.01	2.43	83	1024
	80	0	0.01	-2.67	17.77	197	1000
	100	0	0	2.36	99.35	333	2104
	200	0	0.01	-679.46	2068.54	1423	3326
	300	0.05	0.52	-2985.96	-408.19	1748	2804
0.4	40	0	0	-0.4	0.08	67	503
	60	0	0	-1.17	1.9	68	671
	80	0	0.01	1.34	19.96	42	494
	100	0	0.01	-10.26	68.16	433	2144
	200	0	0.01	-196.64	2043.44	1316	4549
	300	0	0.01	-2099.43	916.81	1351	2717
0.5	40	0	0	-0.45	0.4	71	585
	60	0	0.01	-0.52	1.63	48	490
	80	0	0.01	-1.54	18.96	149	772
	100	0	0	-10.77	95.47	558	2560
	200	0	0.01	-489.51	1495.5	1457	3434
	300	0	0	-978.76	451.26	974	1584
0.6	40	0	0	-0.39	0.49	78	482
	60	0	0.01	-1.35	1.06	144	1440
	80	0	0.01	-1.54	21.37	237	1976
	100	0	0	-7.35	81.4	489	1688
	200	0	0.01	-461.65	2107.72	1285	3428
	300	0	0	-2243.49	853.16	1264	2304
0.7	40	0	0	-0.35	0.42	57	439
	60	0	0	-0.96	1.95	100	1282
	80	0	0	-3.83	19.42	202	1129
	100	0	0	-10.92	113.96	476	1332
	200	0	0.13	-547.39	1786.61	1483	3667
	300	0.01	0.09	-834.5	1164.47	1157	3531
0.8	40	0	0	-0.24	0.25	61	612
	60	0	0	-0.98	1.79	68	649
	80	0	0.01	-6.87	7.43	312	2012
	100	0	0	-16.14	52.61	486	1827
	200	0.01	0.35	-112.72	1847.14	1050	2550
	300	0	0.05	-1189.3	2137.28	943	2228

Table 4.2: Solution quality and running time for CPLEX solving instances with pseudo-dominance

Size	Min. Filtered (%)	Avg. Filtered (%)	Max. Filtered (%)
40	0.00	2.25	7.50
60	0.00	3.06	8.33
80	0.00	3.00	7.50
100	1.00	3.37	6.00
200	1.50	3.13	5.00
300	1.33	3.23	5.00
400	1.25	3.15	5.25
500	0.80	2.93	5.40
600	2.17	2.88	3.67
700	2.00	2.99	4.43

Table 4.4: Proportion of vertices removed from instances by the filtering step

However, since our work focuses on a heuristic approach and the pseudo-dominance condition, we propose to remove more vertices by sorting them in decreasing order of their dominance set size and by removing the first 10%, 15%, and 20% of the most dominated queries.

vertices removed	Size	Δ_{avg}	Δ_{max}	ΔT_{avg} (s)	ΔT_{max} (s)	#Nodes$_{avg}$	#Nodes$_{max}$
10%	40	0	0	-0.1	0.84	70	510
	60	0	0.01	-0.38	3.14	151	2486
	80	0	0.01	0.65	22	117	1048
	100	0	0.01	0.03	101.14	444	1745
	200	0	0.01	160.61	2252	1462	4563
	300	0.24	1.83	1493.71	5770.97	1017	2420
15%	40	0	0	0.07	0.76	41	360
	60	0	0	-0.51	2.53	182	1347
	80	0	0.01	1.67	21.97	65	634
	100	0	0	2.89	119.44	430	1200
	200	0	0.01	528.63	3786.35	1201	3919
	300	0.18	1.14	**1409.66**	6512.28	1220	1924
20%	40	0.27	2.88	238.87	3586.31	86528	1329708
	60	0.06	1.01	0.19	2.38	13	138
	80	0.02	0.28	1.51	18.71	198	1202
	100	0	0.06	4.22	122.9	562	2021
	200	0.02	0.32	516.67	2800.96	1037	3229
	300	0.03	0.17	**148.61**	6510.58	1089	1966

Table 4.5: Solution quality and run time for CPLEX solving Filtered instances

We report in Table 4.5 the number of nodes #Nodes explored by CPLEX, and the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value (computed without filtering). Finally, we report the average and

maximum time gained ΔT_{avg} and ΔT_{max} compared to running CPLEX on unfiltered instances. Similarly to previous experiments, negative values represent a longer time. The results in Table 4.5 show that it is beneficial to remove more queries than only the 3% that would have been removed by using the bound N on optimal solution size. Removing 10% to 15% of queries yields a significant improvement in the running time (of more than 20 minutes on larger instances) without deteriorating too much the solution quality: the deviation is at most 1.83% with respect to optimal solutions. Although deviations remain very low with 20% of vertices filtered, we gain less time on larger instances than with 15% filtered. We will therefore use a 15% filtering on all further experiments.

4.5.3 Initial heuristics

We now evaluate the quality of the initial heuristics h-tsp and h-ks. We compare their running times along with their deviation to the optimal solutions computed by CPLEX. We conduct this series of experiments on instances of the family *f1* and *f2*. Those sets of instances respectively constitute Knapsack-like and TSP-like instances. Thus, we expect *h-ks* to perform better on *f1*, and *h-tsp* to perform better on *f2*. Additionally, given the results of the previous experiments, we propose to test the effect of the filtering approach on the heuristics. We report the average and maximum deviation, respectively Δ_{avg} and Δ_{max}, from the optimal solution value along with the average running time for each heuristic in Table 4.6.

As we expected, results presented in Table 4.6 show a clear advantage of *h-ks* for the knapsack-like instances (family *f1*). The average deviation of *h-ks* remains mostly under 1%, with its running time well under one second. Meanwhile, *h-tsp* produces solutions with a deviation of 5-6% with a slightly higher running time. When examining the results on instances from family *f2*, however, we note the superiority of *h-tsp* over *h-ks* is marginal. Yet, results in Table 4.7 show that *h-tsp* produces better solutions than *h-ks* in some cases. However, due to the low cost of running those heuristics on most instance types, we still believe it is better to run both and pick the best result.

Now considering the filtering step, we report the result of running both heuristics on filtered instances in Table 4.8. The filtering seems to marginally degrade the heuristic solutions in a few scenarios. We also note a reduction in the running time of *h-tsp* on family *f1* as a side effect of the filtering. However, with the initial heuristics already very fast with small instances, we propose to reserve it only for larger instances (>100 queries). In the next experiments, we evaluate the pre-filtering combined with our matheuristics.

4.5.4 Comparison of the matheuristics with optimal solutions

We now compare the four matheuristics by evaluating their deviation to the optimal solution when they are all given a 10-minute time budget. We conduct this experiment on both instances of the family *f3* and *f4* and with and without the

Familly	Algorithm	Size	Δ_{avg}	Δ_{max}	Avg. running time (s)
f1	*h-ks*	40	0.27	1.87	0.01
f1	*h-ks*	60	0.00	0.00	0.01
f1	*h-ks*	80	0.05	0.63	0.01
f1	*h-ks*	100	0.7	4.81	0.01
f1	*h-ks*	200	0.28	1.34	0.01
f1	*h-ks*	300	0.09	0.35	0.01
f1	*h-tsp*	40	6.9	12.49	0.02
f1	*h-tsp*	60	4.34	8.44	0.12
f1	*h-tsp*	80	5.80	9.55	0.54
f1	*h-tsp*	100	6.03	7.61	1.03
f1	*h-tsp*	200	5.74	8.32	5.42
f1	*h-tsp*	300	5.73	8.40	27.03
f2	*h-ks*	40	0.6	1.82	0.01
f2	*h-ks*	60	0.29	1.16	0.01
f2	*h-ks*	80	0.22	0.87	0.01
f2	*h-ks*	100	0.32	0.69	0.01
f2	*h-ks*	200	0.11	0.33	0.01
f2	*h-ks*	300	0.07	0.22	0.01
f2	*h-tsp*	40	0.32	1.82	0.02
f2	*h-tsp*	60	0.36	1.11	0.05
f2	*h-tsp*	80	0.19	0.84	0.12
f2	*h-tsp*	100	0.26	0.69	0.31
f2	*h-tsp*	200	0.13	0.33	1.05
f2	*h-tsp*	300	0.04	0.22	1.31

Table 4.6: Run time and quality of solutions produced by initialization heuristics on unfiltered instances

Family	*h-ks*	*h-tsp*	tie
f1	175	0	5
f2	24	32	124
f3	23	3	154
f4	152	18	10

Table 4.7: Number of instances on which a heuristic performs better than the other, with the number of ties.

Table 4.8: Running time and quality of solutions produced by initialization heuristics on filtered instances

Familly	Algorithm	Size	Δ_{avg}	Δ_{max}	Avg. running time
f1	*h-ks*	40	0.27	1.87	0.01
f1	*h-ks*	60	0	0	0.01
f1	*h-ks*	80	0.05	0.63	0.01
f1	*h-ks*	100	0.7	4.81	0.01
f1	*h-ks*	200	0.28	1.34	0.01
f1	*h-ks*	300	0.09	0.35	0.01
f1	*h-tsp*	40	5.19	10.39	0.01
f1	*h-tsp*	60	4.18	8.43	0.02
f1	*h-tsp*	80	4.29	9.2	0.06
f1	*h-tsp*	100	5.21	9.89	0.12
f1	*h-tsp*	200	5.17	7.01	0.80
f1	*h-tsp*	300	4.49	6.62	3.51
f2	*h-ks*	40	0.6	1.82	0.01
f2	*h-ks*	60	0.29	1.16	0.01
f2	*h-ks*	80	0.3	1.61	0.01
f2	*h-ks*	100	0.39	2	0.01
f2	*h-ks*	200	0.22	1.67	0.01
f2	*h-ks*	300	0.09	0.22	0.01
f2	*h-tsp*	40	0.27	1.82	0.10
f2	*h-tsp*	60	0.32	1.11	0.46
f2	*h-tsp*	80	0.27	1.61	0.76
f2	*h-tsp*	100	0.37	2	0.64
f2	*h-tsp*	200	0.21	1.67	0.95
f2	*h-tsp*	300	0.07	0.22	2.33

Algorithm	m_{iter}	t_{iter}	w	k	h
vpls-det	5	120	15	7	N.A.
vpls-random	7	90	20	10	N.A.
lb-y	7	90	N.A.	N.A.	15
lb-yx	5	120	N.A.	N.A.	50

Table 4.9: Best parameters for the four matheuristics given our preliminary experiments

filtering step. For each matheuristic, we run preliminary tests to find an efficient combination of its parameters, and the obtained results can be found in Table 4.9. Note that for each instance, the matheuristics are given the best initial solution from either *h-tsp* or *h-ks*.

In Table 4.10, we report the results of matheuristics running on unfiltered instances, the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value along with the average running time for each matheuristic. For uncorrelated instances (*f3*) the four algorithms provide extremely low deviations with *vpls-det* appearing to slightly outperform the others. However, this advantage does not appear to hold for instances of family *f4*. With those correlated instances, it appears that *lb-yx* performs significantly better with an average deviation under 5% on the 300 node instances. As explained in Section 4.2, we only report in Table 4.11 results using the filtering approach on instances of family *f4*. With this filtering step, we observe better results on large instances as matheuristics do not reach their time limit. The deviations are overall extremely low. As for the fastest algorithm, *vpls-random* manages an average of 66 seconds on instances of 300 vertices.

4.5.5 Performance on larger instances

As a final experiment, we propose to evaluate the capabilities of the matheuristics with the filtering step to solve larger instances of the TAP, where the mathematical solver runtime exceeds an hour. We compare the performance of the four matheuristics with a 10-minute time limit on the largest instances of family *f4*. We report δ^I_{avg} the deviation to the initial (*h-tsp* or *h-ks*) solution for each matheuristic in Table 4.12.

The results in Table 4.12 show that all matheuristics are equally able to improve the solutions provided by the initialization heuristics. A 2% to 4% improvement is achieved depending on the size of the instances. The largest instances only improved by 2%. This shows the ability of the matheuristic to scale beyond the MIP solver with instances of up to 700 nodes. However, beyond that point, the two initial heuristics remain the only viable options.

Family	Algorithm	Size	Δ_{avg}	Δ_{max}	Avg. time (s)
f3	*vpls-det*	40	0	0	0.97
f3	*vpls-det*	60	0	0	2.71
f3	*vpls-det*	80	0	0	8.26
f3	*vpls-det*	100	0	0	14.65
f3	*vpls-det*	200	0	0.02	143.54
f3	*vpls-det*	300	0.19	0.5	514.97
f3	*vpls-random*	40	0	0	0.92
f3	*vpls-random*	60	0	0	2.68
f3	*vpls-random*	80	0	0	8.51
f3	*vpls-random*	100	0	0	16.91
f3	*vpls-random*	200	0	0.02	172.30
f3	*vpls-random*	300	0.25	0.53	451.24
f3	*lb-y*	40	0	0	1.14
f3	*lb-y*	60	0	0	3.19
f3	*lb-y*	80	0	0	9.36
f3	*lb-y*	100	0	0	18.90
f3	*lb-y*	200	0.04	0.57	173.03
f3	*lb-y*	300	0.26	0.53	588.67
f3	*lb-yx*	40	0	0	1.01
f3	*lb-yx*	60	0	0	2.84
f3	*lb-yx*	80	0	0	8.83
f3	*lb-yx*	100	0	0	17.27
f3	*lb-yx*	200	0.01	0.22	195.25
f3	*lb-yx*	300	0.26	0.53	555.34
f4	*vpls-det*	40	0	0	0.71
f4	*vpls-det*	60	0	0.01	1.70
f4	*vpls-det*	80	0	0	2.78
f4	*vpls-det*	100	0	0.1	10.59
f4	*vpls-det*	200	0.02	0.05	68.17
f4	*vpls-det*	300	18.16	99.85	211.16
f4	*vpls-random*	40	0	0	0.65
f4	*vpls-random*	60	0	0.01	1.54
f4	*vpls-random*	80	0	0	3.73
f4	*vpls-random*	100	0	0.1	12.53
f4	*vpls-random*	200	0.01	0.08	85.17
f4	*vpls-random*	300	9.16	99.8	217.31
f4	*lb-y*	40	0	0	1.04
f4	*lb-y*	60	0	0.01	2.69
f4	*lb-y*	80	0	0	6.23
f4	*lb-y*	100	0	0	21.69
f4	*lb-y*	200	0.45	12.86	210.37
f4	*lb-y*	300	6.18	34.96	323.70
f4	*lb-yx*	40	0	0	0.89
f4	*lb-yx*	60	0	0.01	2.25
f4	*lb-yx*	80	0	0	5.14
f4	*lb-yx*	100	0	0	19.70
f4	*lb-yx*	200	0.23	6.69	198.66
f4	*lb-yx*	300	4.81	26.86	227.97

Table 4.10: Solution quality and running time of matheuristics on instances of families f3 and f4 (without filtering)

Algorithm	#vertices	Δ_{avg}	Δ_{max}	Avg. time (s)
vpls-det	40	0	0	0.57
vpls-det	60	0	0.01	1.30
vpls-det	80	0	0.01	3.12
vpls-det	100	0	0.05	9.07
vpls-det	200	0.02	0.07	49.87
vpls-det	300	0.01	0.04	103.94
vpls-random	40	0	0	0.56
vpls-random	60	0	0.01	1.28
vpls-random	80	0	0.01	3.20
vpls-random	100	0	0	10.84
vpls-random	200	0	0.04	94.95
vpls-random	300	0.01	0.04	66.75
lb-y	40	0	0	0.82
lb-y	60	0	0.01	2.13
lb-y	80	0	0.01	5.43
lb-y	100	0	0	19.17
lb-y	200	0	0.03	213.71
lb-y	300	0.01	0.04	274.33
lb-yx	40	0	0	0.68
lb-yx	60	0	0.01	1.80
lb-yx	80	0	0.01	5.21
lb-yx	100	0	0	16.81
lb-yx	200	0	0.01	147.67
lb-yx	300	0	0.04	204.48

Table 4.11: Solution quality and running time of matheuristics on filtered instances of family f4

4.6 Conclusion

In this chapter, we studied the solutions to the Traveling Analyst Problem (TAP) using: a MIP solver, simple and effective heuristics based on Knapsack and TSP heuristics, and finally, matheuristics based on the VPLS and Local Branching methods. We investigated possible techniques to accelerate solution of the MIP further using filtering and additional constraints based on pseudo-dominance. Both heuristics and matheuristic showed capabilities to produce high-quality solutions in less than 10 minutes. In this chapter, we have shown that these various tools are adapted to a wide variety of TAP instances and contexts. In the last two chapters of this thesis, we will demonstrate their use in appropriate scenarios. In Chapter 5, we show the MIP used in small instances and the heuristics in instances up to millions of nodes. Finally, in Chapter 6, we present a novel method that uses the potential of matheuristics for fast, high-quality solutions.

#vertices	Algorithm	δ^I_{avg}
400	*vpls-det*	4.04
400	*vpls-random*	4.05
400	*lb-y*	4.05
400	*lb-yx*	4.01
500	*vpls-det*	3.28
500	*vpls-random*	3.28
500	*lb-y*	3.28
500	*lb-yx*	3.28
600	*vpls-det*	2.88
600	*vpls-random*	2.88
600	*lb-y*	2.88
600	*lb-yx*	2.88
700	*vpls-det*	2.57
700	*vpls-random*	2.57
700	*lb-y*	2.57
700	*lb-yx*	2.57

Table 4.12: Relative deviation over initial solutions for instances of family *f4*

Chapter 5

Applications

Contents

5.1 Introduction

In this section, we highlight three applications of the TAP problem and the methods used to solve them. Two applications are related to the EDA field ([Chanson *et al.* 2022a, Chanson *et al.* 2022b]) while the third one concerns personalized lifelong pathways ([Chanson *et al.* 2021]). We briefly discuss the scope and contributions of [Chanson *et al.* 2021] and [Chanson *et al.* 2022b] and dedicate the rest of this chapter to a major contribution of this thesis ([Chanson *et al.* 2022a]).

Generating personalized EDA notebooks Our first contribution to the EDA field ([Chanson *et al.* 2022b]) proposes a way to use existing EDA notebooks produced by the user community on platforms such as Kaggle and creating personalized EDA exploration by re-using their base components (code or text cells). We compute interestingness of a cell is determined by its similarity to the user's profile. In practice, given the nature of the notebook as a tool for sharing code, we noticed the user's intent was often clearly explained in the form of comments. Thus we used the TF-IDF[1] method for these textual elements, creating vectors for the profile and cells. The interestingness of a cell is then computed by a cosine similarity between its vector and the user profile vector. The same vectors are used to compute the distance between cells using the cosine distance function ([Singhal & Google 2001]). Finally, the time is set to a constant to control the length of the notebooks generated. We conducted a series of preliminary experiments to construct notebooks from ten thousand cells about a heart disease dataset[2]. The use of machine learning to learn interest and distance meant the construction of the instance was orders of magnitude longer than solving it heuristically. Although the produced[3] notebooks seem somewhat coherent, we noted several issues: notably, some cells may require variables or libraries that have not been initialized in the crowdsourced notebook. This could be solved by analyzing the source and prepending the necessary code to the cells in the crowdsourced notebook.

Lifelong pathways for RSA benificiaries Our second application of TAP does not concern EDA sessions. In [Chanson *et al.* 2021], we aim to construct lifelong pathways. These pathways are sequences of actions pertaining to health, social, or professional aspects for fulfilling a lifelong personal project. This work focuses on the pathways undertaken by French RSA[4] beneficiaries to get a job. The nature of this project required that the interestingness measure be personalized to the beneficiary profile. We used machine learning to identify the relevance of each action to a beneficiary expressed as the probability a beneficiary with a similar profile would have undertaken this action. In this work, we explored the option of a learned distance function. We used the method described in [Xing *et al.* 2002] to learn a generalized Mahalanobis distance based on existing pairs of actions in completed pathways. As for time, we used the real-world time (in days) required for an action (e.g., "Get a driver's license" 30 days). This was extracted from data of previous beneficiaries having completed their pathway to employment. In this work, we were able to directly solve the MIP (see Section 4.2) as the number of actions was limited.

[1] TF-IDF (Term Frequency - Inverse Document Frequency) as an early statistical language model ([Salton & Buckley 1988]) that detects frequent words across documents and establishes their relative importance in every document. It outputs vectors that can then be used to compare documents based on the prevalence of frequent words in them.

[2] https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

[3] Samples available here: https://github.com/Blobfish-LIFAT/NotebookCrowdsourcing/tree/master/output/notebooks

[4] https://en.wikipedia.org/wiki/Revenu_de_solidaritÃ_active

We used previously gathered data on finished pathways to compare our algorithm to human performance. We could reconstruct similar pathways in most cases (0.9 of F_1 measure) in less than 5 minutes. In this context, using a MIP has a major advantage. It permits the user to add domain-specific constraints at will, such as precedence constraints between actions.

5.2 Generation of sequences of comparison queries

We dedicate the rest of this chapter to introducing a major contribution of this thesis. In this work, we contribute to the field of automatic generation of data exploration sessions ([Tang *et al.* 2017, El *et al.* 2020]). 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. First, we assume that the dataset consists of one table, imported into a Relational DataBase Management System (RDBMS), for which the user only has to distinguish between numeric attributes and categorical attributes before starting to query it with SQL.

This works focuses on comparison queries, we refer the reader to Section 3.4 for a formal definition. We assume that the user is interested in a sequence of comparison queries, that we call a *comparison notebook* in this chapter. This sequence of maximum interestingness, minimum overall distance, and minimum execution time will be constructed by solving the TAP. Consistently with prior approaches ([Zgraggen *et al.* 2018, Tang *et al.* 2017]), we consider that insights should be validated using appropriate statistical tests. Therefore we integrate statistical testing and filtering of non-significant insights in our work.

An overview of our approach is presented in Figure 5.1. When 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 evidence of insights. Then, only those aggregate queries that are evidence 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 the distance between them.

This chapter is organized as follows. Section 5.3 introduces hypothesis queries and insights. Section 5.4 formalizes the problem while Section 5.5 presents our solution schemes. Section 5.6 presents the tests we have done. Finally, Section 5.7 concludes and draws some research perspectives.

5.3 Comparison queries, hypothesis queries, and insights

This section presents our logical framework. Section 5.3.1 defines comparison queries and notebooks, and Section 5.3.2 defines insights and hypothesis queries. Finally,

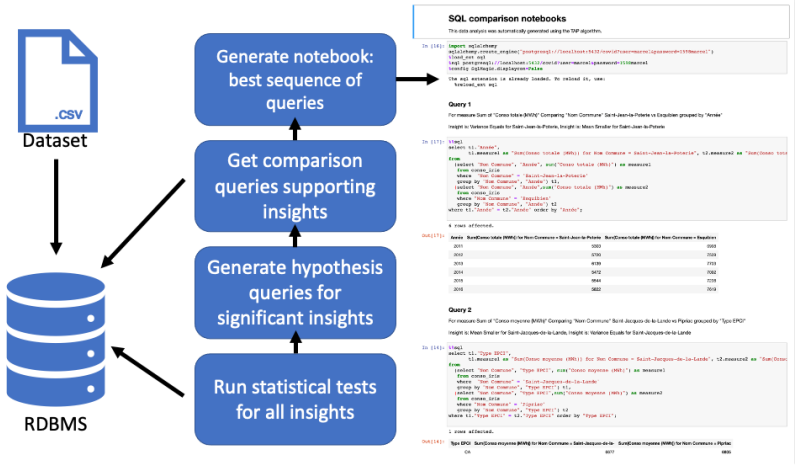

Figure 5.1: Overview of the approach

Section 5.3.3 introduces insight credibility and a transitivity relation over insights.

5.3.1 Comparison queries

This work introduced and used the comparison query; we refer the reader to Section 3.4 for a definition and properties of comparison queries. Note that in this chapter, we assume there is no functional dependency between categorical attributes.[5] 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. In this context, this notebook is simply a TAP solution.

5.3.2 Insights and hypothesis queries

Consistently with previous characterizations of insights in EDA [Tang *et al.* 2017, Zgraggen *et al.* 2018], we see *insights* over a dataset as declarations such as "On average there were more COVID cases in May compared 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 the 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.

[5] 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.

Definition 7 (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 (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 an hypothesis over relation R.

Definition 8 (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 of such query is given in Figure 5.2.

Definition 9 (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 an hypothesis that can be tested.

Definition 10 (Query supporting an insight)**.** *Given an 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. Also, note 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 $(A, B, val, val', M, agg)$

Table 5.1: 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 5.2: A hypothesis query postulating insight $avg(April) < avg(May)$.

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 are evidence of 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 considered test and the null hypothesis depend on the insight type (see Table 5.1).

Definition 11 (Insight significance)**.** *Let $\pi_{\tau \to hypothesis}(\sigma_p(q))$ be an 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 4.
An example of hypothesis query for the comparison query of Figure 3.2 is given in Figure 5.2. 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 3.2 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 $\lvert\mu_X - \mu_Y\rvert$ is applied over R, showing that $avg(May) - avg(April) = 55.79$. The p-value gives the significance of the insight as

the probability of observing 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.

5.3.3 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.* 2018], 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 a 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 12 (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 $\left(1 - \frac{credibility(i)}{|Q^i|}\right)$ 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 pruning 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 that 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.

5.4 Comparison notebooks generation

We now define the problem of generating sequences of comparison insights. Consistently with EDA (e.g., [El *et al.* 2020]), our objective is to generate compelling exploratory sessions, specifically coherent sequences of comparison queries showing significant insights. We model this problem as an $\varepsilon-$constrained TAP and discuss

the interestingness, cost, and distance used in this application in this section. Finally, we discuss the construction of the set of comparison queries that constitute the TAP instance.

5.4.1 Interestingness, cost, and distance

Interestingness Following [Tang *et al.* 2017, Zgraggen *et al.* 2018, Marcel *et al.* 2019, El *et al.* 2020], 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, for (iv), we use a conciseness measure in the spirit of that introduced in [El *et al.* 2020].

Definition 13 (Interestingness of a query)**.** *Let q be a comparison query and I_q be the set of insights supported by q. The interestingness of query q is $interest(q) =$*

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

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 enables us to "spread" the ideal ratio. A geometric intuition of the function behavior is given in Figure 5.3 (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 must 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 a vector of query parts $q = (A, B, val, val', M, agg)$ that can be extracted from the text of the query. Following [Aligon *et al.* 2014], the weights represent the importance of the query parts in the transition from one comparison

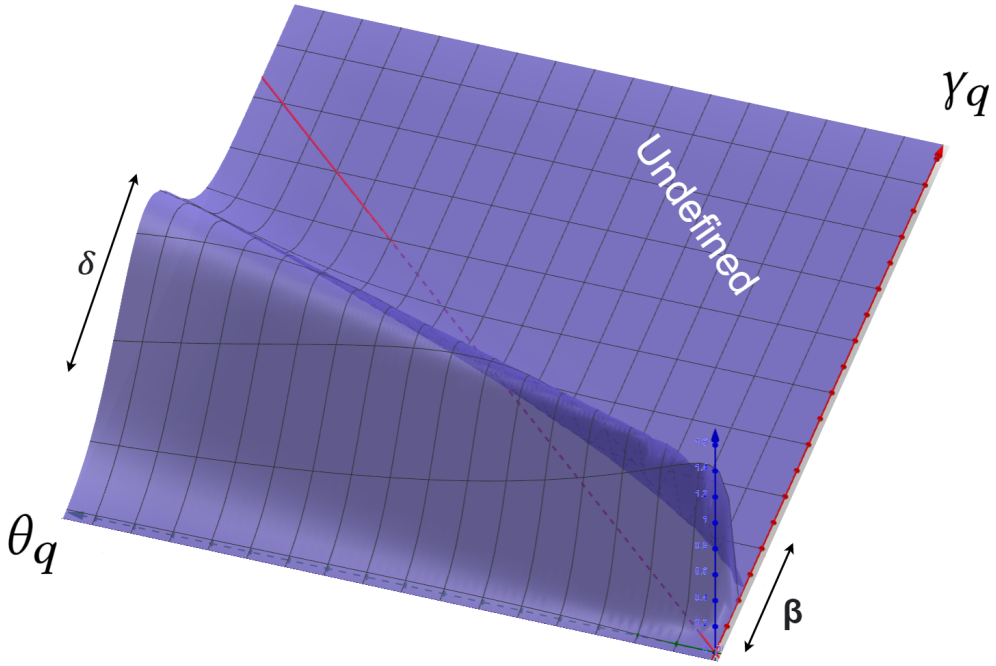

Figure 5.3: Illustration of the *conciseness* function

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 5.6 which confirms this intuition (see Figure 5.4). In this case, only interestingness and distance have an impact on the computation of an optimal 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.

5.4.2 Generating the set of comparison queries

The generation of the set Q of comparison queries over a dataset R is performed by Algorithm 6. 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 an hypothesis query for each possible grouping attribute and aggregation function (line 5), and next checking if this hypothesis query supports the insight (line 8). Finally, for the sets of comparison queries that evidence the

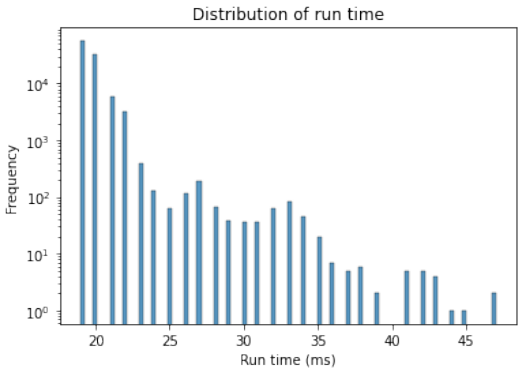

Figure 5.4: Distribution of comparison queries run times

same insights (line 15), only the ones maximizing interestingness are kept (line 16).

Algorithm 6 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.

5.5 Optimizing comparison notebook generation

Generating comparison queries with Algorithm 6 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 a very large number of insights and comparison queries, we implemented three types of optimizations: (i) optimize statistical tests, (ii) minimize the number of queries to send to the DBMS, and (iii) use a heuristic to approximate the TAP.

5.5.1 Optimizing statistical tests

5.5.1.1 Using permutation testing

For hypothesis testing we use resampling, due to its advantages over parametric testing [Zgraggen *et al.* 2018]: 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. Since many statistical tests are performed we are likely to encounter spurious insights due to the multiple comparison problem [Zgraggen *et al.* 2018]. This issue is however well studied by statisticians ([Benjamini 2010]), and we can use a p-value correction method to ensure the false discovery likelihood remains consistent with the usual 5%. We use the [Benjamini & Hochberg 1995] false discovery rate correction method to correct p-values, setting the false discovery rate at 5%.

Algorithm 6 Comparison query generation

Require: a relation R
Ensure: a set of comparison queries over R

$Q \leftarrow \emptyset$
for each insight $i = (M, B, val, val', p)$ of type τ over R **do**
  compute $sig(i)$ ▷ perform statistical test
  if $sig(i) > 0.95$ **then** ▷ i is significant
    for each $A \neq B$ of R and function agg **do**
      $q \leftarrow (A, B, val, val', M, agg)$
      $h \leftarrow \pi_{\tau \rightarrow hypothesis}(\sigma_p(q))$
      if $h \vdash i$ **then** ▷ h supports i
        $Q \leftarrow Q \cup \{q\}$
      end if
    end for
  end if
end for
for each $q = (A, B, val, val', M, agg) \in Q$ **do**
  $Q^A \leftarrow \{q' \in Q | (C, B, val, val', M, agg)$ with $C \neq A\}$
  $Q \leftarrow Q \setminus Q^A \cup \{argmax_{q' \in Q^A} interest(q')\}$
end for
return Q

5.5.1.2 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.

5.5.2 Reducing the number of queries

Algorithm 6 (i) requires, 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 $\{M_1, \ldots, M_m\}$. For generating comparison queries, Algorithm 6 (ii) requires evaluating 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 evaluating $n(n-1)/2$ 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.

Table 5.2: 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

5.5.2.1 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 the DBMS. To reduce the number of hypothesis queries to evaluate, we remark that we only need 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 1) 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.

5.5.2.2 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 [Ma *et al.* 2021] and presented in Algorithm 7. 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., $G = 2^A \setminus \{A_1\} \setminus \ldots \setminus \{A_n\}$ (Algorithm 7, 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 (Algorithm 7, 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 $O(|U|\times log|G|)$ [Young 2016]. 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 memory.

Algorithm 7 Finding the best set of group-by sets

Require: a relation R with n categorical attributes $A = \{A_1, \ldots, A_n\}$
Ensure: a set of group-by sets over R with minimal memory footprint covering all pairs of categorical attributes

```
G ← 2^A
G ← G \ {g ∈ G | |g| = 1}
U ← {g ∈ G | |g| = 2}
for each group-by set g of G do
    q ← γ_g(R)
    Estimate the size of q
end for
G ← Solve the weighted set cover problem for G, U
return G
```

Table 5.3: Implementations

Name	Generation of Q	Solving TAP
Naive-exact	Algo. 6 + bounding	MIP
Naive-approx	Algo. 6 + bounding	h-KS
WSC-approx	Algo. 7	h-KS
WSC-unb-approx	Algo. 7 + unbalanced-sampling	h-KS
WSC-rand-approx	Algo. 7 + random-sampling	h-KS

5.6 Experimental results

5.6.1 Experimental setup

The real datasets for our tests are described in Table 5.2. The tiny Vaccine dataset[6] consists of country-level COVID-19 vaccination data as of June 2021. The ENEDIS dataset[7] is about electric consumption in France by location, year, consumption category, and commercial sector. The Flights dataset[8] consists of one year of flight arrival and departure details for all commercial flights within the USA.

The implementations used in the tests are described in Table 5.3, 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 5.5. Naive-exact uses the naive Algorithm 6 and the optimization of Section 5.5.2.1 for generating the set of comparison queries, and then we solve the MIP of the TAP (see Section 4.2). In Naive-approx, approximating TAP is done with our heuristic h-KS. This heuristic is also used for implementations of WSC-approx, WSC-unb-approx, and WSC-rand-approx, which use Algorithm 7 of Section 5.5.2.2 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-

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

Table 5.4: 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

sampling for WSC-rand-approx. In all our tests, the parameters of the conciseness function (see Section 5.4) 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[9]. 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 GHz Intel Xeon 5118 12-core, 24 logical processors and 377GB 2666 MHz of DDR4 main memory.

5.6.2 Exact resolution of the TAP

This first test aims to answer the following questions: 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 default settings (notably single-threaded), with a timeout set to one hour, on 30 instances of equal size, for a given number of queries ($\varepsilon_t = 25$) in the solution. We report the average time by size in Table 5.4.

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.

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

5.6.3 Scalability

Since, as shown above, exact solution is only doable for small datasets, this next test aims at answering the question: how well does the implementation of a reasonable heuristic 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 solution time, varying the dataset size and the number of queries expected in the solution, i.e., ε_t (see Section 5.4), 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.

5.6.3.1 Adjusting sample size

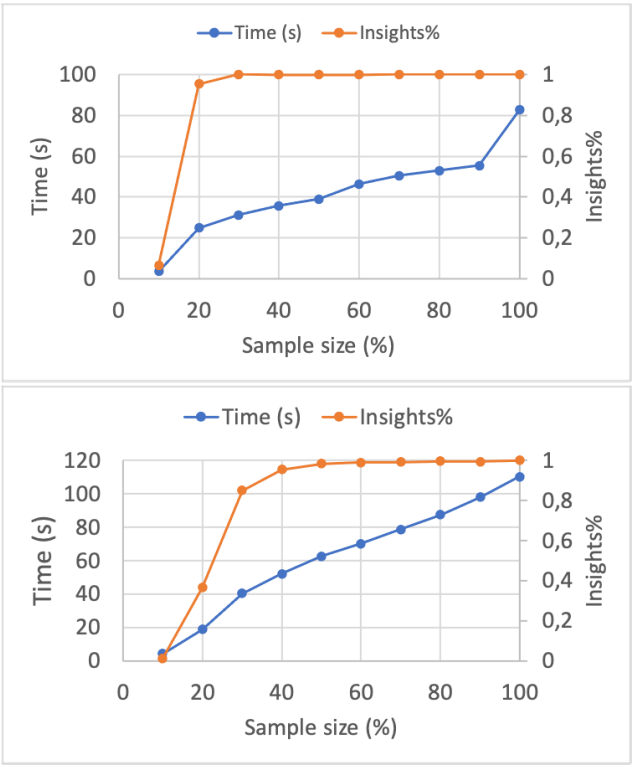

Figure 5.5: Adjusting sample size for WSC-unb-approx (top) and WSC-rand-approx (bottom)

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 5.5, 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 preserve more insights at lower sampling rates.

5.6.3.2 Runtime breakdown

For this test, we ran the 5 implementations on the Enedis dataset.

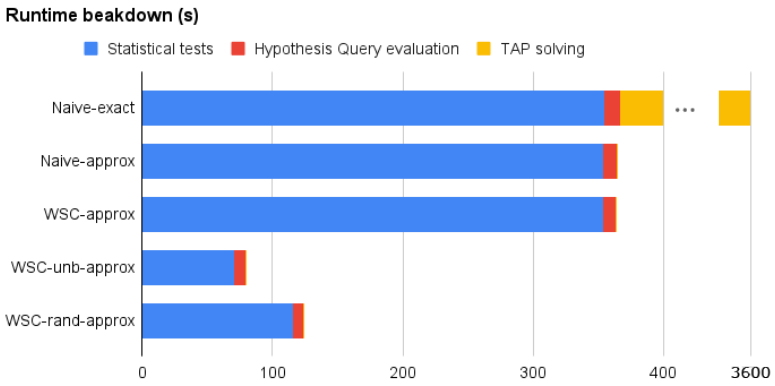

Figure 5.6: Runtimes breakdown on the ENEDIS dataset

The result is shown in Figure 5.6, the average runtime breakdown by implementation.

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. We note that solving TAP heuristically is negligible when compared to the instance construction. Finally, we see that Algorithm 7 has only little impact on the hypothesis query evaluation, which can be explained by the small number of categorical attributes in the dataset.

5.6.3.3 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

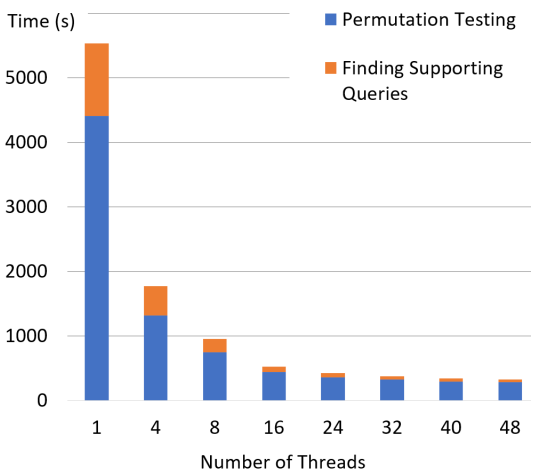

Figure 5.7: Impact of multi-threading on the generation of Q

WSC-approx on the ENEDIS dataset, with 1 to 48 threads, and report the runtime for steps (i) and (ii) in Figure 5.7. 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.

5.6.3.4 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 run WSC-unb-approx and WSC-rand-approx on Flights, testing different sample sizes in $\{5\%, 10\%, 20\%, 30\%\}$. The results are shown in Figure 5.8.

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 detected insights , for both implementations, is greater than 100. This is due to the extreme reduction in the dataset size using aggressive sampling factors during statistical tests: some detected insights 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 5.4.1), being computed on the complete dataset, could be used to control the weight given to these spurious insights.

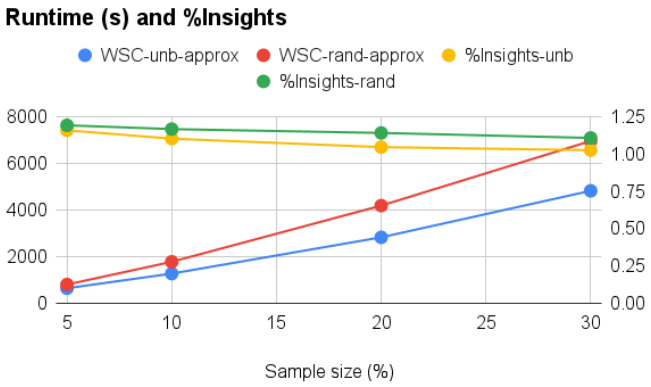

Figure 5.8: Runtime and % of insights on the Flights dataset

Table 5.5: Average deviation to optimal solution objective

#Queries	Deviation (± std.)
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 %

5.6.4 Quality of approximate solutions

This test aims at answering the question: how degraded are the approximate solutions of the TAP compared to the optimal ones? For this test, we used h-KS to find approximate solutions of the TAP.

Our first experiment uses the same artificial datasets as the ones used in Section 5.6.2, 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 the quality of the solution, we compute z, i.e., the sum of interestingness of the queries in the solutions. We show in Table 5.5 $((cplex.z - \text{h-KS}.z)/cplex.z) \times 100$, i.e., the deviation between $cplex.z$, the quality of solutions found by CPLEX when solving the MIP, and h-KS.z, the quality of solutions found by h-KS.

The deviation remains very low in general, indicating that h-KS is effective when considering query interestingness. Deviations are greater for small instances and decrease with larger instances. This is expected since, for smaller instances, as the 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

Table 5.6: Deviation to optimal solution

#Queries	Recall (Algorithm 1)	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 5.7: Notebook generators for user tests

Name	Sampling	Sample size	Interestingness	Solving TAP
Naive-exact	-	100%	full	CPLEX
WSC-approx	-	100%	full	Algo. 1
WSC-approx-sig	-	100%	sig. only	Algo. 1
WSC-approx-sig-cred	-	100%	sig. and cred. only	Algo. 1
WSC-unb-approx	unbalanced	10%	full	Algo. 1
WSC-rand-approx	random	10%	full	Algo. 1

recall in Table 5.6.

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 5.6. 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 1. As shown in Table 5.6, Algorithm 1 is steadily around 2.5 to 3 times better than the Baseline.

5.6.5 Human evaluation

This last test aims at answering the following 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 5.7. The versions of Naive-exact, WSC-approx, WSC-unb-approx, and WSC-rand-approx are as described in Table 5.3. 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

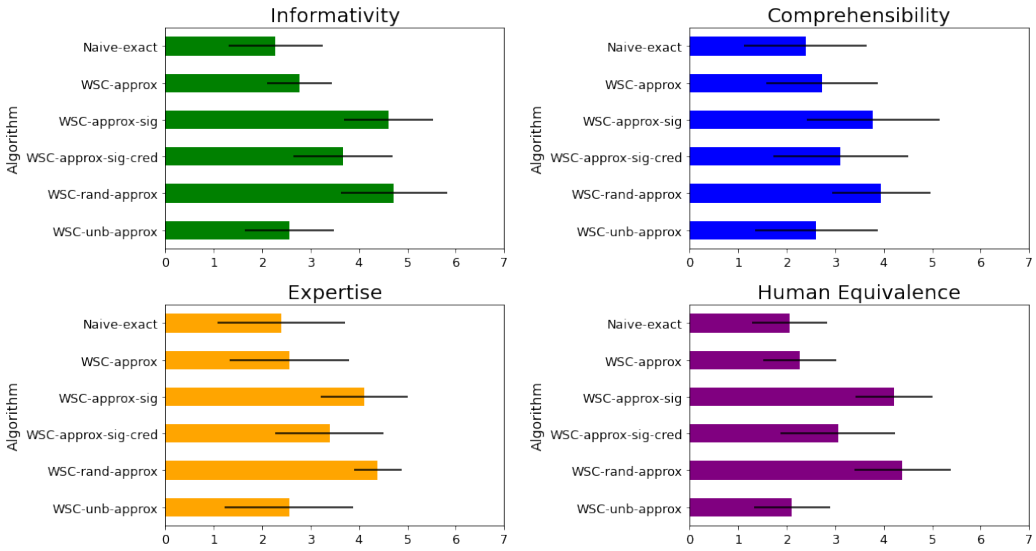

Figure 5.9: Qualitative human evaluation

conciseness.

The generated notebooks were deployed on Jupyter[10] and were presented to the volunteers, insisting on the fact that the notebooks have to 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 [El *et al.* 2020]: (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 5.9. 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 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 generated notebook can still be deemed informative. The low scores received by Naive-exact tend to indicate that an exact solution 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

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

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 5.6 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 solution, and secondly, it cannot be said that sampling impacts how human evaluate the notebooks.

5.7 Conclusion

This work 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.

The results on runtime, especially Figure 5.6 on the breakdown of runtime between TAP instance construction and solving, highlight the need for a novel approach. Indeed, although we were able to construct and use various optimization schemes to accelerate instance generation, it still takes the vast majority of the runtime. We speculate the most efficient approach would be to only partially generate the TAP instance.

The next and final chapter of this thesis addresses the solution of the TAP without generating the complete instance.

Chapter 6

Results on non-enumerable space

Contents

6.1 Motivation

In Section 4, we demonstrated that given a small or medium database, we can generate all comparison queries, compute a complex interestingness measure and solve a large TAP instance within an hour of CPU time. However, this was only possible at the cost of several optimizations at every step of the process. Therefore, a different approach is needed to tackle larger databases with more than a few million comparison queries. As some of those optimizations rely on storing data in RAM it's expected scaling them to larger databases may not be possible. Our approach's run time was mainly affected by generating the instance, as seen in Figure 5.6. Therefore, our best course of action is to design a process that does not require us to generate all comparison queries and evaluate their cost and interest. Hopefully, we can identify methods for solving similar problems in both the OR and Database research communities.

Context A similar problem to the intractable enumeration of all comparison queries arises in the OR community. Indeed some problems, such as (but not limited to) vehicle routing problems ([Feillet 2010]), may be formulated with an exponential number of variables but few constraints. This makes using a classic LP or MIP

solver intractable for even small instances. To solve this issue, Column Generation (CG) only considers a subset of the problem's variable at a time. This smaller problem is often called Restricted Master Problem (Restricted Master Problem (RMP)) ([Winston 2022]). CG then relies on iteratively solving a sub-problem called Pricing Problem (PP); its role is to select variables that are added to the RMP in order to improve its current best solution. This method can be repeated until no improving variable can be found to obtain an optimal solution. However, it can also be used as a heuristic by limiting the number of iterations or using approximation when solving the sub-problem.

In the database community, a fairly recent way to automate EDA is through the use of Reinforcement Learning (RL). In [Bar El *et al.* 2019], Bar EL *et al.* propose to use RL to train an agent, performing a small set of actions sequentially. The agent only considers the next query accessible from its current query, such as a more specific filter or a different aggregation function. This effectively limits the search space to the neighborhood of the starting query. Similarly, in [Personnaz *et al.* 2022], Personnaz *et al.* created an RL-augmented EDA tool dedicated to the exploration of data from the Sloan Sky Survey [Blanton *et al.* 2017]. They let the user decide when and how to use the RL agent they trained, either in a semi-automated fashion or by allowing it to construct a complete session. Unfortunately, the algorithms proposed in those first attempts at using RL for solving the automated EDA problem exhibit the behavior of typical greedy algorithms. They suffer from pitfalls like greedy algorithms, such as severe sensibility to initialization and lack of backtracking capability.

Compared to RL-based approaches such as [Bar El *et al.* 2019, Personnaz *et al.* 2022], column generation does not construct its final solution iteratively. Instead, it produces a new one each time the RMP is solved. As a result, RL-based approaches may be locked into early 'bad choices' that cannot be undone, like greedy algorithms. This is notable in the results presented by [Personnaz *et al.* 2022], where the RL-based approaches only marginally outperform the author's greedy heuristic at a tremendous training time cost (100 hours on server-grade hardware). Furthermore, in all RL works on EDA we identified training seems to be necessary on a per-dataset basis. We choose to pursue our use of OR tools as it seems to offer a more efficient use of computing resources.

By transposing column generation to our context, we propose an approach with a TAP problem restricted to a subset of all comparison queries and a problem that constructs additional queries improving the TAP's solution. Since our approach is only inspired by column generation, and to avoid confusion, we do not use the RMP and PP terminology. We name the restricted TAP primal and its query construction sub-problem dual.

6.2 A query generation Method

Refresher on definitions We refer the reader to Section 4.2 for a refresher on the TAP MIP as it will be used as a basis for new models described in this chapter.

Furthermore, as the method described in this chapter constructs comparison queries, we point the reader to Section 3.4 for their definition and properties.

Query generation (QG) draws heavily from column generation in its design. The general principle described in Algorithm 8 is similar to many column generation methods. First, a small subset of queries, commonly denoted by Q', is sampled from the complete set of all queries Q (Line 1). Then, an iterative process solves a dual problem that aims to generate a query that improves the optimal solution of the primal over Q' (Line 2-4). This query is then added to Q'. After the final iteration, the primal is solved over Q' (now containing the subsequently generated queries) to return a solution (Line 6). Since the size of Q' is small, any method previously described (see Chapter 4), such as heuristics, matheuristics, or MIP solver, can be used to obtain the final TAP solution.

Algorithm 8 General query generation algorithm

Require: A relation R, epsilon constraints values (ε_t and ε_d), and an initialization strategy init that draws queries from Q. n_{pool} and n_{gen} integers, respectively, the size of the starting pool and the number of queries to generate using the dual MIP.

Ensure: A solution to the TAP of service-time at most ε_t and overall distance at most ε_d.

1: $Q' \leftarrow \text{init}(n_{pool}, Q)$.
2: **while** $|Q'| < n_{pool} + n_{gen}$ **do**
3: $\quad q' \leftarrow$ solve dual MIP
4: $\quad Q' \leftarrow Q' \cup \{q'\}$
5: **end while**
6: Solve TAP on Q'

In the context of query generation, the dual problem needs to construct a comparison query that can be added to a solution to the original TAP problem over Q'. This immediately raises an issue, indeed, when solving TAP either using a MIP or a heuristic, it was typically assumed that all queries were known beforehand. This allowed the use of any method found in the literature to compute interest, time, or distance. Indeed when considering the problem's formulation, they were constants. However, the dual problem needs to incorporate methods to estimate all three as part of its formulation. To avoid falling back to the untractable enumeration of all queries, the estimators need to be independent of query evaluation. This yields our first research question: **(RQ1) How can linear estimators dependant only on instance statistics and query text be expressed to compute time, interest, and distances of queries?**

Furthermore, a method must be devised to create a starting pool of queries to populate Q' before the dual can be used to generate new queries. Indeed, as previously implied, the dual is usually formulated to improve an existing solution over an existing set. This yields a cold start problem. While the obvious solution of randomly creating a starting pool of queries is always possible, intuitively, a good

starting set of solutions is expected to lead to a more efficient algorithm. Obviously, this process has to be as fast as possible to increase the number of dual iterations. This constitutes our second research question: **(RQ2) How can we generate a pool of relevant queries to bootstrap the query generation process ?**

In the query generation process, the dual problem is considered to be the most crucial aspect. It involves creating a new comparison query that takes into account the solution to the TAP on Q', as well as its interest, time, and distances as computed by estimators. The dual should be solved quickly to enable more iterations. Moreover, it is important to ensure that any approximations made while solving the dual do not result in generating queries that do not contribute to enhancing the solution. This yields our final research question: **(RQ3) How can we efficiently generate a query that improves a TAP solution ?**

6.3 Query evaluation and generation

In this section, we first address **RQ1** and present methods to estimate time, interest, and distance between queries. We then move to **RQ2** and elaborate on several algorithms to bootstrap the query generation process. Finally, we address **RQ3** and present two MIP models to generate improving queries.

6.3.1 Estimating interest, time and distance

As previously mentioned enumerating Q is intractable for large databases. This has consequences on the way we compute interest and time for queries. Measures that rely on the results of the queries, such as the one used in [Chanson *et al.* 2022a], and many others from the literature (see Section 2.5), are not usable in this context. Likewise, querying the DBMS planner for every estimated execution time would defeat the advantages of not enumerating Q. Therefore, we must choose an interestingness measure and query time estimate that can be computed from only the query text itself, basic statistics on the database, and information about the DBMS storage organization. Given we aim to construct a MIP model based on those estimators, we require linear analytical expressions of those estimators. This doesn't mean that machine learning, as we previously used (Chapter 5) is not possible, but the model learned should remain linear. In this chapter, however, as we are interested in evaluating the potential of query generation, we will not use any machine learning methods that would add an extra layer of noise.

Interest Estimator As discussed in Section 2.5, many interestingness measures in the literature rely on query results. The reliance on the result (or an approximation thereof) of the queries was the main factor driving the running time of our previous approach (Section 5). The team behind Quickinsight ([Ding *et al.* 2019]) uses the market share to construct part of its interestingness. According to the authors, this so-called *impact* can be computed using, among others, the COUNT aggregation function. Using this particular aggregation function, *impact* represents

the number of tuples used to compute the comparison query result, divided by the total number of tuples in the database. Notably, this is similar to the confidence measure used in rule mining [Geng & Hamilton 2006]. Given a comparison query $\tau_A((\gamma_{A,agg(\alpha)\rightarrow left}(\sigma_{B=val}(R))) \bowtie (\gamma_{A,agg(\alpha)\rightarrow right}(\sigma_{B=val'}(R))))$ its *impact* can be computed directly as $(\gamma_{COUNT(*)}(\sigma_{B=val \vee B=val'}(R)))/\gamma_{COUNT(*)}(R)$. This *impact* measure can also be computed using only query text and the selectivity estimates of the DBMS ([Garcia-Molina *et al.* 2009]). Indeed given sel^B_{val} and $sel^B_{val'}$ the estimators. It can be expressed as a linear function which is presented along with the dual model in Section 6.3.3 equation (6.3).

Time Estimator Given the nature of the comparison query pattern the logical execution plan for different queries is always the same Figure (6.1.a). Likewise, when no secondary data structures are present, and the relation is not sorted on disk, optimizers will tend to produce the same plan for many queries. Figure (6.1.b) shows a typical plan produced by the Postgresql optimizer for executing a comparison query. Note that most of the time is spent performing the table scan, which leaves very little influence on other choices such as the join operator. Significant changes to the physical plan can occur only when sorting or secondary data structures such as materialized views or indexes are present. The presence of such a structure should be expected in many cases. Indeed both traditional DBMS ([Chaudhuri *et al.* 1999, Dageville *et al.* 2004]) and cloud-based solutions ([Das *et al.* 2019]) incorporate automated systems to create such structures with limited (or no) user input.

Considering this peculiarity, we propose to create a simple estimator that reflects the redundant optional structures in place that could significantly influence the execution time. Therefore, we assume that query execution time is only dependent on which attributes are used for selection and group by. Thus, for each relation, we consider a series of constants that describes the execution time of a query for every (ordered) pair of attributes $p \in \{(A_1, A_2) | A_1 > A_2, A_1 \in \mathcal{A}, A_2 \in \mathcal{A}\}$. These constants can be quickly estimated by running a few queries for each pair of attributes or by analysis of the DBMS's catalog of secondary structures.

Distances We use the distance proposed in Section 5.4.1 based on Aligon's Work ([Aligon *et al.* 2014]) only relies on query text and thus can be used directly for our QG approach. This distance is based on the differences between query parts, with the weights representing 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.

6.3.2 Generation of the starting pool

Our method introduces a cold start problem; although the dual can generate hundreds of queries, its formulation relies on simultaneously constructing a new query and a TAP solution including this new query. Therefore, we need an initial set of

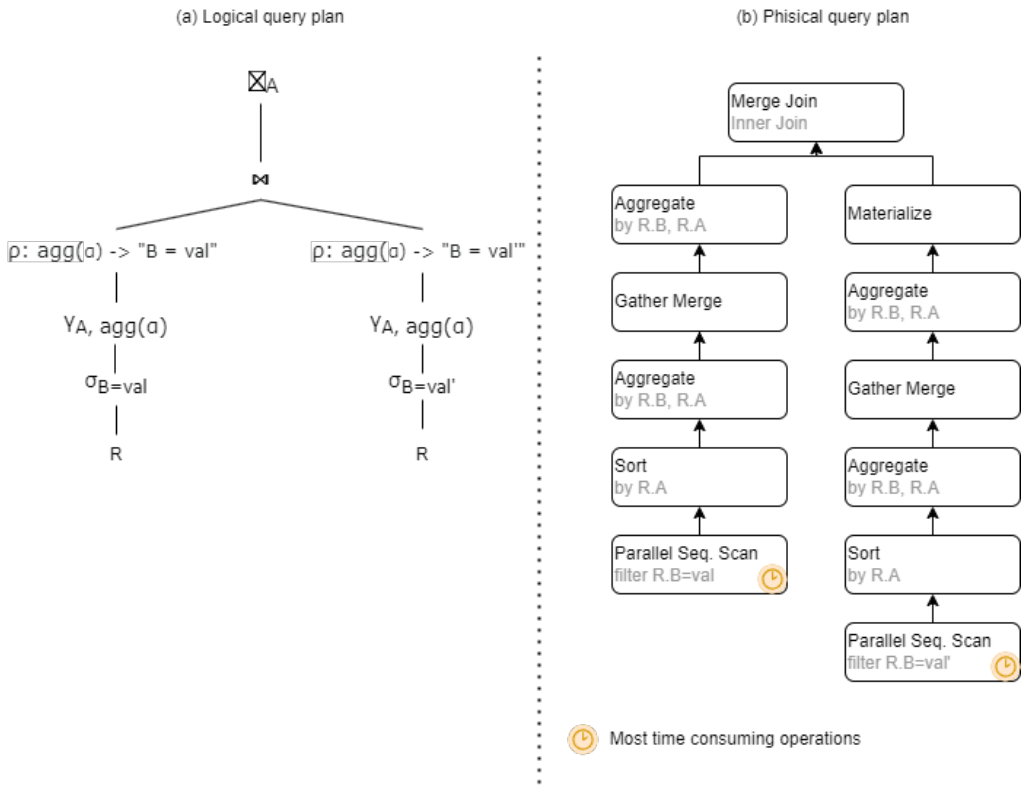

Figure 6.1: Logical and physical execution plan for a typical comparison query

queries to build the first few solutions.The queries in this pool should ideally be diverse to avoid steering the process to a local minimum. They should also produce high-quality TAP solutions when considered as whole instances and solved using methods previously introduced in Chapter 4. Indeed, starting from good solutions may require fewer iterations of the dual to reach the desired quality of the final solution. Furthermore, in pursuing a high-quality final solution, intuitively, high-interest and low-cost queries will improve the quality of the solution.

We propose several methods, divided into two separate categories, called top-down and bottom-up methods. Top-down methods take a large sample of randomly generated queries and reduce it to the required starting pool size. On the other hand, bottom-up methods start with one seed query and generate the rest of the pool. This seed query can be randomly generated or issued by the user. Although this is making them closer to the semi-automatic approach presented in [Bar El *et al.* 2019], unlike in [Bar El *et al.* 2019] this seed query may later be discarded and not included in the final TAP solution.

Bottom-up We propose two methods that perform a bottom-up generation of the starting pool; they can both be formulated as MIP, similar to Dual-MIP. The first method *intensification* generates a set of queries whose distance is the smallest possible to all the previous queries. This distance takes into account: interestingness, time, and the hamming distance used in our previous work ([Chanson *et al.* 2022a]). This MIP relies on the same variables as the dual MIP described in Section 6.3.3 but with a reduced set of constraints (namely constraints (6.4), (6.5), (6.6), (6.7), (6.8), (6.11) and (6.12)) and a specific objective :

$$min \sum_{i=1}^{n} \left(\sum_{k=1}^{n_a} \psi_k(1 - S_{i,k}) + (1 - \psi_k) S_{i,k} + \left| I_i - \sum_{k=1}^{n_a} \sum_{j=1}^{D(A_i)} \beta_{kj}(l_{kj} + r_{kj}) \right| + \left| T_i - \sum_{k=1}^{n_a} \sum_{j=1, k \neq j}^{n_a} \omega_{kj} \phi_{kj} \right| \right) \tag{6.1}$$

Where T_i and I_i are the known interestingness and time (constants) of already generated queries in the pool. The linearization of the absolute values is omitted for clarity. A detailed description of the intensification method is provided in Algorithm 9

The second approach is called the *diversification* method. It aims to create queries as distinct as possible from the seed and the previous queries. It is identical to the previously described intensification approach except for its objective being reversed to a maximization.

Both *diversification* and *intensification* methods can also be combined to create a further two bottom-up approaches described in Figure 6.2.

Top-down Top-down methods rely on pruning a random sample (larger by at least an order of magnitude than the desired starting pool size) of queries to a much

Algorithm 9 Intensification

Require: A seed query q_0, the size of the desired starting pool n_{pool}.
Ensure: A set of queries of size n_{pool} similar to q_0.

```
1: P ← {q_0}
2: while |P| < n_pool do
3:     q' ← solve intensification MIP
4:     P ← P ∪ {q'}
5: end while
6: return P
```

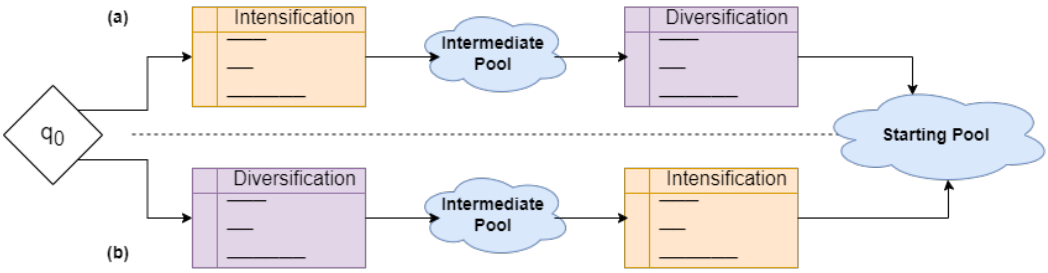

Figure 6.2: Overview of the combined bottom-up starting pool generation methods

smaller starting pool. The first approach, Random-Then-Sort, sorts the random sample according to each query's interest/time ratio; the starting pool is then drawn from the top-k queries according to this ratio in descending order. This approach is simple and as fast as the sorting algorithm used. It should guarantee a set of very interesting, and fast queries; however, it ignores other factors, such as distance or the diversity of queries. The second approach, Random-KS, consists of applying the h-KS heuristic (Algorithm 1) on the random sample and using the queries of the solution found as the starting pool. If required, this can be repeated on the rest of the sample as shown in Algorithm 10.

The last approach, called Hybrid-KMeans++, adds to Random-KS a trick used for centroid initialization in Kmeans clustering. The detailed algorithm is described in Algorithm 11. We first obtain a heuristic solution on the sample like the Random-KS approach (Line 1). Next, the rest of the queries are generated to introduce diversity and coverage in the starting pool (Lines 4-13). This is done by using the same probabilistic distance-based sampling used in the KMeans++ algorithm [Arthur & Vassilvitskii 2007] (Line 5-9). This algorithm was initially designed to avoid the tendency of the Kmeans heuristic to fall into local optima when starting with random cluster centroids.

Algorithm 10 Random-KS

Require: The size of the desired starting pool n_{pool}. A random query sample, S, drawn uniformly from Q. The epsilon constraint values (ε_t and ε_d).

Ensure: A set of queries (starting pool) of size n_{pool}.

$s_0 \leftarrow$ h-KS$(S, \varepsilon_t, \varepsilon_d)$ (see Algorithm 1)
$P \leftarrow set(s_0)$
$S \leftarrow S \setminus P$
while $|P| < n_{pool}$ **do**
    $s_i \leftarrow$ h-KS$(S, \varepsilon_t, \varepsilon_d)$
    $P \leftarrow P \cup set(s_i)$
    $S \leftarrow S \setminus \{s_i\}$
end while
return P

Algorithm 11 Hybrid-KMeans++

Require: The size of the desired starting pool n_{pool}. A random query sample, S, drawn uniformly from Q. The epsilon constraint values (ε_t and ε_d). A total order over S allowing sequence-like access to the set elements.

Ensure: A set of queries (starting pool) of size n_{pool}.

$s_0 \leftarrow$ h-KS$(S, \varepsilon_t, \varepsilon_d)$ (see Algorithm 1)
$P \leftarrow set(s_0)$
$S \leftarrow S \setminus P$
while $|P| < n_{pool}$ **do**
    for $q \in S$ **do**
        $w[i] \leftarrow \arg\min_j d(q, q'), q' \in P$ ▷ Distance to closest query in S
    end for
    $w \leftarrow norm(w)$ ▷ Normalize w s.t. $w_i = w_i / \sum_{j=1}^{|S|} w_j$
    $k \leftarrow Random([1, |S|], w)$ ▷ Draw random integer weighted by w
    $q_s \leftarrow S[k]$
    $S \leftarrow S \setminus \{q_s\}$
    $P \leftarrow P \cup \{q_s\}$
end while
return P

6.3.3 Improving Query Generation

According to **RQ3**, the dual needs to construct a query improving the current solution to the primal on a given subset $Q' \subset Q$. The simplest approach would be to solve the primal on Q' using a MIP solver. For maximum flexibility, we choose to express the dual problem as a MIP that incorporates variables and constraints from the primal in its formulation. The dual will construct simultaneously a new query and a solution to TAP over Q' containing this query. A set of binary variables will describe the new query while the TAP objective and epsilon constraints (see Section

4.2) are modified to consider the new query as part of the TAP solution. We choose to 'force' the inclusion of the new query in the solution. This avoids the need for two linearizations and provides a potential way to stop the cycle of dual iterations if we observe adding a query has degraded the previous solution. We name this model Dual-MIP.

We also provide an alternative model where the primal variables are considered as a linear relaxation and sub-tour elimination constraints are removed, we name this alternative model Dual-LP. The main goal of the Dual-LP is to provide a model faster to solve, enabling more iterations within a given time limit.

6.3.3.1 Dual-MIP and Dual-LP models

Data

$S_{i,j} \in \{0,1\}$, with $i \in 1...|\mathcal{A}|, j \in 1...n$ denotes the presence of the i^{th} categorical attribute in the selection of q_j.

$\beta_{i,j} \in \{0,1\}$, with $i \in 1...|\mathcal{A}|, j \in 1...D(A_i)$ the selectivity of the j^{th} value of the categorical attribute A_i over R.

ϕ_{ij}, with $i \in 1...|\mathcal{A}|, j \in 1...|\mathcal{A}|, i \neq j$ denotes the estimated time to execute the query: $\tau_{A_i}((\gamma_{A_i,agg(\alpha)\rightarrow left}(\sigma_{A_j=val}(R))) \bowtie (\gamma_{A_i,agg(\alpha)\rightarrow right}(\sigma_{A_j=val'}(R))))$. Where the i^{th} categorical attribute is used for aggregation and the j^{th} is used for selection.

Variables

$$\alpha_i = \begin{cases} 1 & \text{if } M_i \text{ is the measure used in} \\ & \text{the comparison query} \\ 0 & \text{otherwise} \end{cases} , \forall i \in 1..|\mathcal{M}|$$

$$\Gamma_i = \begin{cases} 1 & \text{if } A_i \text{ is a categorical attribute used in the} \\ & \text{group by set of the comparison query} \\ 0 & \text{otherwise} \end{cases} , \forall i \in 1...|\mathcal{A}|$$

$$\psi_i = \begin{cases} 1 & \text{if } A_i \text{ is a categorical attribute used} \\ & \text{for selection in the comparison query} \\ 0 & \text{otherwise} \end{cases} , \forall i \in 1...|\mathcal{A}|$$

$$z_i = \begin{cases} 1 & \text{if } agg_i \text{ is the aggregation function used} \\ & \text{in the comparison query} \\ 0 & \text{otherwise} \end{cases} , \forall i \in 1...|\mathcal{F}|$$

$$l_{ij} = \begin{cases} 1 & \text{if constant } j \text{ in the active} \\ & \text{domain of } A_i, \textit{adom}(A_i) \text{ is} \\ & \text{used as } \textit{val} \\ 0 & \text{otherwise} \end{cases}, \forall i \in 1...|\mathcal{A}|, \forall j \in 1...D(A_i)$$

$$r_{ij} = \begin{cases} 1 & \text{if constant } j \text{ in the active} \\ & \text{domain of } A_i, \textit{adom}(A_i) \text{ is} \\ & \text{used as } \textit{val}' \\ 0 & \text{otherwise} \end{cases}, \forall i \in 1...|\mathcal{A}|, \forall j \in 1...D(A_i)$$

$$\omega_{ij} = \begin{cases} 1 & \text{if } A_i \text{ is a categorical attribute used in the} \\ & \text{group by set of the comparison query} \\ & \textbf{and } A_j \text{ is a categorical attribute used} \\ & \text{for selection in the comparison query} \\ 0 & \text{otherwise} \end{cases}, \forall i \in 1...|\mathcal{A}|, \forall j \in 1...|\mathcal{A}|, i \neq j$$

Finally, in addition to the Variables mentioned in this section, we add the variables from the main TAP problem Section 4.2. Their domain remained unchanged for Dual-MIP while they are relaxed in $[0, 1]$ for Dual-LP.

Objective

$$max \sum_{i=1}^{n} p_i y_i + I \tag{6.2}$$

With I an estimation of the interest (market share) of the new query:

$$I = \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1}^{D(A_i)} \beta_{ij}(l_{ij} + r_{ij}) \tag{6.3}$$

Constraints First, several constraints are needed to ensure we respect the comparison query pattern defined in Section 3.4.

$$\sum_{i=1}^{|\mathcal{F}|} z_i = 1 \tag{6.4}$$

Constraint (6.4) ensures the query has only one aggregation function.

$$\sum_{i=1}^{|\mathcal{M}|} \alpha_i = 1 \tag{6.5}$$

Constraint (6.5) ensures we select one measure.

$$\sum_{i=1}^{|\mathcal{A}|} \Gamma_i = 1 \tag{6.6}$$

Constraint (6.6) ensures that one attribute is present in the group key.

$$\sum_{i=1}^{|\mathcal{A}|} \psi_i = 1 \tag{6.7}$$

Constraint (6.7) ensures that one attribute is used for selection.

$$\psi_i + \Gamma_i \leq 1, \forall i \in 1..|\mathcal{A}| \tag{6.8}$$

Constraints (6.8) ensure there is no overlap between attributes used for selection and group by operations.

$$\sum_{j=1, i \neq j}^{|\mathcal{A}|} \omega_{ij} \geq \Gamma_i, \forall i \in 1..|\mathcal{A}| \tag{6.9}$$

$$\sum_{j=1, i \neq j}^{|\mathcal{A}|} \omega_{ji} \geq \psi_i, \forall i \in 1..|\mathcal{A}| \tag{6.10}$$

Constraints (6.9) and (6.10) ensure that ω_{ij} is tied to variables describing the selection and group by operations of the pattern.

$$\sum_{j=1}^{D(A_i)} l_{ij} = \sum_{j=1}^{D(A_i)} r_{ij} \leq \psi_i, \forall i \in 1..|\mathcal{A}| \tag{6.11}$$

$$l_{ij} + r_{ij} \leq 1, \forall i \in 1...|\mathcal{A}|, \forall j \in 1...D(A_i) \tag{6.12}$$

Constraints (6.11) and (6.12) ensure that if an attribute is used for selection we take one constant from its domain as val and another as val'.

We also add the constraints from the original TAP model (see Section 4.2), except sub-tours elimination constraints (4.7) only present in Dual-MIP. The epsilon constraints (4.2), (4.3), from the original model are modified to incorporate estimators for query time and distances. First, we introduce the estimator described in Section 6.3.1 into the time epsilon-constraint:

$$\sum_{i=1}^{n} t_i y_i + \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1, i \neq j}^{|\mathcal{A}|} \omega_{ij} \phi_{ij} \leq \varepsilon_t \tag{6.13}$$

Next, we modify the distance epsilon-constraint, incorporating the Hamming distance to the new query:

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} d_{i,j}x_{i,j}\\&\quad+\sum_{i=1}^{n} x_{i,n+1}\left(\sum_{k=1}^{|\mathcal{A}|}\psi_k(1-S_{i,k})+(1-\psi_k)S_{i,k}\right)\\&\quad+\sum_{i=1}^{n} x_{n+1,i}\left(\sum_{k=1}^{|\mathcal{A}|}\psi_k(1-S_{i,k})+(1-\psi_k)S_{i,k}\right)\leq\varepsilon_d\end{aligned}\tag{6.14}$$

Finally, for every query previsouly generated or in the starting pool $q_e \in Q'$ a duplicate elimination constraint is added to the model:

$$\begin{aligned}(1-\Gamma_\gamma)+\left(\sum_{i=1,i\neq\gamma}^{|\mathcal{A}|}\Gamma_i\right)+(1-\alpha_\mu)+\left(\sum_{i=1,i\neq\mu}^{|\mathcal{M}|}\alpha_i\right)+(1-l_{\rho,v_1})+(1-r_{\rho,v_2})\\+\sum_{i=1}^{|\mathcal{A}|}\sum_{j=1,\neg(i=\rho\ \&\ j=v_1)}^{D(A_i)} l_{i,j}+\sum_{i=1}^{|\mathcal{A}|}\sum_{j=1,\neg(i=\rho\ \&\ j=v_2)}^{D(A_i)} r_{i,j}\\\leq|\mathcal{A}|+|\mathcal{M}|+2\left(\sum_{i=1}^{|\mathcal{A}|}D(A_i)\right)-1,\forall q_e\in Q\end{aligned}\tag{6.15}$$

Where γ is the index (position w.r.t. the total order over $\mathcal{A}$) of the q_e group by attribute, μ the index of its measure, ρ the index of its selection attribute, and s_l ans s_r the indexes of the attributes values used for both left and right selections.

Linearization of (6.14) As the modified distance epsilon-constraint is non-linear its use in a MIP requires linearization. The following constraints and variables are used in it's it's place:

$$\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} d_{i,j}x_{i,j}+\sum_{i=1}^{n} D_i+D'_i\leq\varepsilon_d\tag{6.16}$$

$$D_i\geq\left(\sum_{k=1}^{|\mathcal{A}|}\psi_k(1-S_{i,k})+(1-\psi_k)S_{i,k}\right)-(1-x_{i,n+1})M,\forall i\in 1...n\tag{6.17}$$

$$D'_i\geq\left(\sum_{k=1}^{|\mathcal{A}|}\psi_k(1-S_{i,k})+(1-\psi_k)S_{i,k}\right)-(1-x_{n+1,i})M,\forall i\in 1...n\tag{6.18}$$

$$D_i \leq x_{i,n+1} M, \forall i \in 1...n \tag{6.19}$$

$$D'_i \leq x_{n+1,i} M, \forall i \in 1...n \tag{6.20}$$

With $D_i \in \mathbb{N}$ and $M = 2(|\mathcal{A}|+|\mathcal{M}|+1) + 1$.

6.3.3.2 Symmetry elimination constraints

In this section, we discuss possible symmetries in the dual and how we address them. Indeed, symmetries in a MIP may reduce the performance of a mathematical solver. For example, one symmetry that can be removed is directly linked to the query pattern.

Consider $q_1 = \tau_A((\gamma_{A,agg(\alpha)\rightarrow left}(\sigma_{B_1=v_1}(R))) \bowtie (\gamma_{A,agg(\alpha)\rightarrow right}(\sigma_{B_1=v_2}(R))))$ and $q_2 = \tau_A((\gamma_{A,agg(\alpha)\rightarrow left}(\sigma_{B_1=v_2}(R))) \bowtie (\gamma_{A,agg(\alpha)\rightarrow right}(\sigma_{B_1=v_1}(R))))$. Although syntactically different queries, they are the same query. Only differing in the presentation of their result. Those queries were previously eliminated during instance generation in our previous works ([Chanson *et al.* 2022a]).

To this extent, we propose two methods to eliminate symmetric queries; the first method relies on a large set of constraints that relies on the lexical order of attribute values to only allow queries $q_i = \tau_A((\gamma_{A,agg(\alpha)\rightarrow left}(\sigma_{B_1=v}(R))) \bowtie (\gamma_{A,agg(\alpha)\rightarrow right}(\sigma_{B_1=v'}(R))))$ such that $v \leq v'$.

$$\sum_{i=1}^{k} \sum_{j=1}^{D(A_i)} l_{ij} <= \sum_{i=1}^{k} \sum_{j=1}^{D(A_i)} r_{ij}, \forall k \in 1..n \tag{6.21}$$

$$\sum_{j=1}^{k} l_{ij} <= \sum_{j=1}^{k} r_{ij} + \left(1 - \sum_{l=1}^{D(A_i)} r_{il}\right), \forall k \in 1..D(A_i), \forall i \in 1..|\mathcal{A}| \tag{6.22}$$

The second method relies on creating an extra constraint for every query already generated or initially present in the starting pool. This yields a set of constraints very similar to (6.15).

$$\begin{aligned}(1-\Gamma_\gamma) + \left(\sum_{i=1, i\neq\gamma}^{|\mathcal{A}|} \Gamma_i\right) + (1-\alpha_\mu) + \left(\sum_{i=1, i\neq\mu}^{|\mathcal{M}|} \alpha_i\right) + (1-l_{\rho,v_1}) + (1-r_{\rho,v_2}) \\ + \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1, \neg(i=\rho\wedge j=v_1)}^{D(A_i)} r_{i,j} + \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1, \neg(i=\rho\wedge j=v_2)}^{D(A_i)} l_{i,j} \\ \leq |\mathcal{A}|+|\mathcal{M}|+2\left(\sum_{i=1}^{|\mathcal{A}|} D(A_i)\right) - 1, \forall q_e \in Q'\end{aligned} \tag{6.23}$$

Database	*enedis*	*insurance*	*flights*
# Tuples	114 527	12 694 445	5 819 079
# Dimensions	7	13	6
# Measures	2	2	3
$\|Adom\|$ min-max	3-1295	2-931	7-6952
# Comparison Queries	10^5	1.1×10^7	1.22×10^8

Table 6.1: Description of the databases used in query generation experiments

Where γ is the index (position w.r.t. the total order over $\mathcal{A}$) of the q_e group by attribute, μ the index of its measure, ρ the index of its selection attribute, and s_l ans s_r the indexes of the attribute values used for both left and right selections.

Although with the second approach, the number of constraints increases at each iteration of the query generation, the overall number of constraints remains relatively small in practice (only a few hundred). In contrast, the first set of constraints may be as large as the number of distinct constants in R.

6.4 Experiments

In order to tune and evaluate our approach, we design three series of experiments. Their goal is first to identify the most promising methods to generate starting pools. The second experiment focuses on improving the query generation phase. We aim to evaluate the behavior and performance of the mathematical solver for both the Dual-LP and Dual-MIP improving query generation. And possibly tune the mathematical solver to decrease the iteration time when solving the Dual-LP model. Finally, we evaluate the overall performance of the solution process compared to a baseline (Figure 6.3 (e)) where all queries are generated and the h-KS heuristic is used to solve the TAP (see Section 5.6.3.4).

databases We choose three databases for the experiments. Table 6.1 provides a summary of these database characteristics. They were chosen to represent three typical scenarios: *enedis* is a small database where a complete generation of comparison queries and costly interestingness measures can be used. *flights* where the baseline can still be executed in an hour. And finally *insurance* where the generation of all comparison queries alone may take several hours, making the baseline intractable. We choose three databases for those experiments. Two of them, *enedis* and *flights*, were previously used in this thesis (see Chapter 5). The database called *insurance* is a health insurance database from the United States: it contains data on insurance rates collected as a part of the Medicare program[1].

[1] https://www.kaggle.com/databases/hhs/health-insurance-marketplace

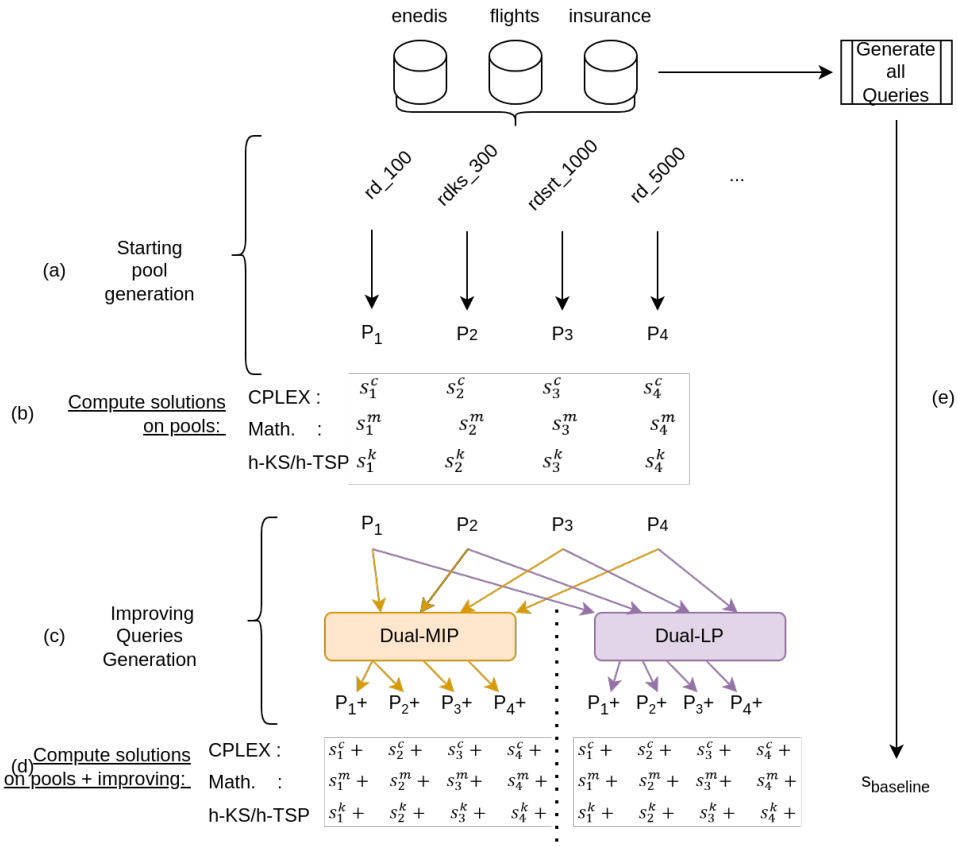

Figure 6.3: Principle of the conducted experiments

Designation	pool size	Generation method
rd_x		Uniform Random
rd_div_x		Diversification (Figure 6.2 (b))
rd_int_x		Intensification (Figure 6.2 (a))
rd_div_int_x	x ∈ {100, 200, 300, 500, 1000, 5000, 10000}	Diversification then intensification (Figure 6.2 (d))
rd_int_div_x		Intensification then diversification (Figure 6.2 (c))
rdsrt_x		Uniform Random (x*100), select top-x ratio (interestingness/time)
rdks_x		Algorithm 10
kmeanspp_		Algorithm 11

Table 6.2: Summary of possible starting pool generations

All experiments are conducted on a Fedora Linux workstation, on a 2.3 GHz Intel Xeon 5118 12-core and 377GB 2666 MHz of DDR4 main memory. Individual test configurations run in a single-core configuration with 16GB of memory unless another configuration is specified. The mathematical solver used for this series of experiments is CPLEX version 20.10.

6.4.1 Starting pool generation methods

This first series of experiments aims to identify which method (see Section 6.3.2) produces the best starting pool. We choose to evaluate the quality of a starting pool by the quality of the TAP solution that can be built upon it. We compare the solutions obtained by solving the TAP MIP, the vpls-sx matheuristic, and the two heuristics (Figure 6.3 (b)) in terms of objective value on different pool sizes and generation methods. We test 8 generation methods summarized in Table 6.2 on seven different sizes of starting pool (see columns 1 and 2 of Table 6.2) on the three databases. One method, rd_x, will serve as a baseline; it is a simple random query generator. Furthermore, to take account of the non-deterministic nature of the algorithms for a given configuration, and size, each algorithm is run 20 times. A timeout of one hour is used on each run.

We present in Tables 6.3, 6.4, 6.5 the deviation to the best-known solution produced by the heuristics[2], matheuristic, and CPLEX on every starting pool configuration for each database. On all databases, we notice that the bottom-up methods are overall not fast enough and appear to time out most of the time. Therefore, we omit the configurations that time out before producing the desired number of queries from the result tables. However, the results in Table 6.3 show that in some limited cases where the starting pool is small and the database is also small (enedis), one of the bottom-up methods seems to produce the overall best solutions. On average bottom-up methods do not perform well. In Table 6.4, the large random sample (5000, 10000 queries) seems to produce overall excellent solutions. However, we also

[2]In order to simplify the result tables, we run both h-KS and h-TSP and only record the best heuristic solution.

	Heuristics			Matheuristic			CPLEX MIP		
Method_Size	min	avg	max	min	avg	max	min	avg	max
rd_100	58.9	64.9	72.9	58.7	64.7	72.8	58.7	64.7	72.8
rd_200	46.5	53.4	60.3	46.4	53.3	60.2	46.4	53.3	60.2
rd_300	37.7	47.2	53.4	37.7	47.1	53.4	37.7	47.1	53.4
rd_500	36.4	37.4	38.4	36.4	37.4	38.4	28.3	37.5	41.9
rd_1000	14.8	20.6	26.5	14.8	20.6	26.5	t. out	t. out	t. out
rd_5000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rd_10000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rdsrt_100	10.6	10.6	10.6	10.2	**10.2**	10.3	10.2	10.2	10.2
rdsrt_200	10.6	10.6	10.6	10.2	**10.2**	10.3	10.2	10.2	10.2
rdsrt_300	10.6	10.6	10.6	10.2	**10.2**	10.3	10.2	10.2	10.2
rdsrt_500	10.5	10.5	10.5	10.2	**10.2**	10.3	10.2	10.2	10.2
rdsrt_1000	10.5	10.5	10.5	10.2	10.3	10.5	t. out	t. out	t. out
rdsrt_5000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rdsrt_10000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rdks_100	10.5	10.5	10.5	10.2	**10.2**	10.2	10.2	10.2	10.2
rdks_200	10.5	10.5	10.5	10.2	**10.2**	10.4	10.2	10.2	10.2
rdks_300	10.5	10.5	10.5	10.2	**10.2**	10.5	10.2	10.2	10.2
rdks_500	10.5	10.5	10.5	10.2	10.5	10.5	10.2	10.2	10.2
rdks_1000	10.5	10.5	10.5	10.5	10.5	10.5	t. out	t. out	t. out
rdks_5000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rdks_10000	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
kmeanspp_100	46.8	49.3	51.4	46.8	49.3	51.4	46.8	49.3	51.4
kmeanspp_200	10.5	10.5	10.5	10.3	10.4	10.5	10.3	10.4	10.5
kmeanspp_300	10.5	10.5	10.5	10.2	**10.2**	10.5	10.2	10.2	10.2
kmeanspp_500	10.5	10.5	10.5	t. out	t. out	t. out	t. out	t. out	t. out
rd_div_100	31.5	32.4	33.4	31.5	32.4	33.4	31.5	32.4	33.4
rd_int_100	4.7	31.6	100.0	4.7	31.6	100.0	4.7	31.6	100.0
rd_div_int_100	29.6	33.7	56.6	29.6	33.7	56.6	29.6	33.7	56.6
rd_int_div_100	**0.0**	19.3	40.1	**0.0**	19.3	40.1	**0.0**	19.2	40.1

Table 6.3: Deviation (%) to best known solution constructed on starting pools (enedis database) by the heuristics, matheuristic, and CPLEX

	Heuristics			Matheuristic			CPLEX MIP		
Method_Size	min	avg	max	min	avg	max	min	avg	max
rd_100	48.4	55.4	61.9	48.4	55.2	61.9	48.4	55.2	61.9
rd_200	33.8	40.8	52.5	33.8	40.7	52.0	33.8	40.7	52.0
rd_300	25.9	32.7	43.0	25.9	32.6	42.7	25.9	32.6	42.7
rd_500	15.9	21.2	27.3	5.9	21.0	27.3	15.9	21.0	26.9
rd_1000	9.3	12.5	14.7	9.3	12.5	14.7	t. out	t. out	t. out
rdsrt_100	0.1	0.7	1.7	0.1	0.7	1.6	0.1	0.7	1.6
rdsrt_200	0.1	0.7	1.9	0.0	0.7	1.9	0.0	0.7	1.9
rdsrt_300	0.2	0.8	1.4	0.2	**0.5**	1.4	0.2	0.7	1.3
rdsrt_500	0.1	0.8	2.4	0.1	0.8	2.4	0.1	0.8	2.3
rdsrt_1000	0.1	0.7	1.9	0.1	0.7	1.9	t. out	t. out	t. out
rdks_100	0.1	0.9	2.5	0.0	0.9	2.5	0.0	0.9	2.3
rdks_200	0.1	0.7	2.3	0.0	0.7	2.3	0.0	0.6	2.2
rdks_300	0.1	0.7	1.8	0.0	**0.6**	1.8	0.1	0.7	1.8
rdks_500	0.1	0.8	1.9	0.1	0.8	1.9	0.1	0.7	1.8
rdks_1000	0.1	0.7	2.5	0.1	0.7	2.5	t. out	t. out	t. out
rdks_5000	0.1	0.8	2.1	t. out	t. out	t. out	t. out	t. out	t. out
kmeanspp_100	18.4	28.2	44.8	18.4	28.2	44.8	18.4	28.2	44.8
kmeanspp_200	0.1	**0.6**	1.4	0.1	**0.6**	1.3	0.1	0.6	1.3
kmeanspp_300	0.1	0.8	2.0	0.1	0.7	2.0	0.1	0.7	2.0
kmeanspp_500	0.6	0.7	0.8	0.6	0.7	0.8	0.2	0.7	1.3

Table 6.4: Deviation (%) to best known solution constructed on starting pools (insurance database) by the heuristics, matheuristic and CPLEX

	Heuristics			Matheuristic			CPLEX MIP		
Method_Size	min	avg	max	min	avg	max	min	avg	max
rd_100	48.2	54.2	60.1	48.1	54.1	60.1	48.1	54.1	60.1
rd_200	31.8	39.1	46.4	31.8	38.9	46.2	31.8	38.9	46.2
rd_300	26.7	32.6	37.5	25.7	32.2	36.9	25.7	32.2	36.9
rd_500	20.0	25.1	33.4	20.0	25.1	33.4	18.6	24.5	32.4
rd_1000	5.6	12.4	17.4	5.6	12.4	17.4	t. out	t. out	t. out
rdsrt_100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdsrt_200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdsrt_300	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdsrt_500	0.0	0.0	0.0	0.0	0.0	0.0	77.8	96.4	98.7
rdsrt_1000	0.0	0.0	0.0	0.0	0.0	0.0	t. out	t. out	t. out
rdks_100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdks_200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdks_300	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rdks_500	0.0	0.0	0.0	0.0	0.0	0.0	t. out	t. out	t. out
rdks_1000	98.6	98.7	99.6	t. out	t. out	t. out	t. out	t. out	t. out
kmeanspp_100	21.8	24.0	25.5	21.8	24.0	25.5	21.8	24.0	25.5
kmeanspp_200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
kmeanspp_300	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
kmeanspp_500	0.0	0.0	0.0	0.0	0.0	0.0	19.0	81.7	98.7

Table 6.5: Deviation (%) to best known solution constructed on starting pools (flights database) by the heuristics, matheuristic and CPLEX

Symmetry Breaking	Constraints (6.21) and (6.22)
	Constraints (6.15)
	CPLEX Heuristics
CPLEX Presolve	0 (disabled)
	1
CPLEX SubMIP Node limit	500
	250
	50
CPLEX MIP Emphasis	0 (balanced)
	1 (feasibility)
	2 (optimality)

Table 6.6: CPLEX parameters (and their values) tested with the Dual-LP model

notice that both CPLEX and the matheuristic time out in this scenario. This hints at those pools being unusable with a similar mathematical model, such as Dual-LP or Dual-MIP, due to their size. Over the three databases, we notice the top-down methods, rdsrt, and rdks, with pool sizes varying from 200 to 500, yield very good solutions whatever the algorithm used to compute a solution next. For kmeanspp, pools of 300 and 500 queries seem to approach rdks and rdsrt performance closely. We propose to use rdsrt and rdks with pools of 200 and 300 queries, along with kmeanspp with pools of 300 queries, in the remainder of this work.

6.4.2 Dual Model solver tuning

As stated previously, the IQG method uses two mixed integer linear programming models, namely Dual-MIP and Dual-LP. In this second series of experiments, we aim to verify our assumption about the performance of both models and tune the solver to obtain the smallest possible iteration time. As preliminary tests have shown its advantage in run-time, we focus our tuning on the Dual-LP model. In addition, the Dual-MIP is used as a reference to ensure the quality of generated queries does not suffer from the gains made on iteration time. We select nine (three for each database) starting pools producing the best solutions (Figure 6.3 (b)). For each database, a pool of 200 queries is generated by rdsrt and rdks (Figure 6.3 (d)). Given a fixed number of 100 iterations, we will evaluate the time per iteration with different values of various parameters and the use of symmetry-breaking constraints (Section 6.3.3). A summary of the tested configurations is given in Table 6.6, and details about specific CPLEX parameters can be found in its documentation ([IBM 2021]). Finally, we compare the generated queries of the fastest configuration to the Dual-MIP approach given a similar computation time (Figure 6.3 (c)).

From the results presented in Table 6.7, we notice that some parameters, such as the sub-MIP node limit, do not have a significant impact when compared to the default configuration. We also notice that depending on the database, some parameters have opposite influences on the time per iteration, notably: turning

Database	configuration	Min. Time	Avg. time	Max. Time
flights	default	383.5	609.1	1270.2
	Cstr. (6.21) & (6.22)	2446.5	2838.2	3185.0
	Cstr. (6.15)	3468.8	3552.6	3636.3
	emp_feas	382.2	784.1	2690.4
	emp_opt	414.7	658.0	2315.5
	node_lim_250	391.2	645.0	1346.1
	node_lim_50	386.3	655.0	1192.4
	presolve_off	321.7	**444.9**	1336.7
enedis	default	769.3	1156.2	1567.1
	Cstr. (6.21) & (6.22)	529.3	**807.0**	922.2
	Cstr. (6.15)	919.4	1421.1	1561.2
	emp_feas	581.3	**1000.7**	1356.3
	emp_opt	769.5	1380.4	2879.5
	node_lim_250	723.3	**1126.0**	1503.1
	node_lim_50	775.4	**1150.5**	1500.7
	presolve_off	751.7	1158.4	1887.4
insurance	default	1231.3	1787.7	2037.2
	Cstr. (6.21) & (6.22)	818.7	**1321.5**	1457.5
	Cstr. (6.15)	1692.2	2852.5	3246.7
	emp_feas	1154.5	**1705.3**	1956.9
	emp_opt	1233.6	1975.1	3308.3
	node_lim_250	1045.5	**1767.1**	2076.4
	node_lim_50	978.6	**1743.3**	2076.3
	presolve_off	1797.1	2567.7	3144.9

Table 6.7: Minimum, average and maximum time taken for 100 iterations of Dual-LP with various configurations on the three databases.

Database	configuration	Min. Time	Avg. time	Max. Time
flights	default	383.5	609.1	1270.2
	best	321.7	444.9	1336.7
enedis	default	769.3	1156.2	1567.1
	best	598.0	615.7	643.8
insurance	default	1231.3	1787.7	2037.2
	best	1601.0	1617.0	1650.0

Table 6.8: Minimum, average and maximum time taken for 100 iterations of Dual-LP with best and default configurations.

off presolve gains time for flights and enedis but is detrimental on insurance; on the other hand, custom symmetry elimination constraints (6.21) and (6.22) and emphasis on feasibility are beneficial for insurance and enedis but not for flights. Based on these findings, we choose to configure CPLEX in its **best** configuration for each database:

- flights, disabling presolve;
- enedis, using our custom symmetry elimination constraints (6.21) and (6.22), a 250 sub-MIP node limit and an emphasis on feasibility;
- insurance, using our custom symmetry elimination constraints (6.21) and (6.22), a 50 sub-MIP node limit and an emphasis on feasibility.

Notably, on insurance, this combination of individually beneficial parameters is not as useful as using our custom symmetry elimination constraints (6.21) and (6.22) alone. We use these configurations for the remainder of this experiment and report their time for 100 iterations compared to the default in Table 6.8.

Finally, to verify our hypothesis that the Dual-LP (with solver tuning) is advantageous when compared to the Dual-MIP, we run both algorithms, on the same starting pools, with a 30-minute time limit. Then we solve the TAP on the starting pool queries and the queries generated by both dual models. Similarly to Section 6.4.1, we use the MIP, the matheuristic, and the heuristics. We repeat the experiment on 20 random starting pools (generated using rdks) for each database.

In Table 6.9, we present the minimum, average, and maximum deviations to the best-known solution for each database on TAP instances generated by Dual-LP and Dual-MIP. We also give the average number of iterations performed by each approach in the 30-minute time budget in Table 6.10.

From Table 6.9, we identify that the best solution for all three databases is found using Dual-LP combined with the matheuristic. The far larger number of iterations reported in Table 6.10 for the Dual-LP method confirms our initial hypothesis that a larger number of iterations is worth the inaccuracies introduced by the relaxation in Dual-LP.

Dual-LP

	Heuristics			Matheuristic			MIP		
Database	min	avg	max	min	avg	max	min	avg	max
enedis	0.13	0.13	0.13	0.0	0.00	0.09	0.0	0.41	3.30
flights	0.00	0.32	0.45	0.0	0.00	0.00	0.0	0.00	0.00
insurance	0.13	0.13	0.13	0.0	0.01	0.02	0.0	0.18	3.82

Dual-MIP

	Heuristics			Matheuristic			MIP		
Database	min	avg	max	min	avg	max	min	avg	max
enedis	2.89	7.89	11.69	2.60	7.83	11.69	2.60	7.88	13.40
flights	0.69	0.98	1.18	0.21	0.48	0.67	0.21	0.48	0.67
insurance	9.07	12.28	17.70	9.07	12.18	17.68	9.07	12.11	17.32

Table 6.9: Deviation (%) to the best-known solution constructed on instance generated with either Dual-MIP or Dual-LP by the heuristics, the matheuristic, and the MIP

Database	nb. iterations Dual-MIP	nb. iterations Dual-LP
enedis	19.9 (± 2.1)	143.2 (± 4.4)
flights	20.0 (± 5.4)	63.0 (± 3.2)
insurance	11.8 (± 1.8)	116.7 (± 4.3)

Table 6.10: Average number of iterations by Dual-LP and Dual-MIP in 30 minutes for all databases

database	starting pool	Heurisitcs			Matheuristic			CPLEX MIP			h-KS
		min	avg	max	min	avg	max	min	avg	max	
flights	kmeanspp_300	0.0	2.3	4.5	0.0	1.7	4.1	0.0	1.7	4.1	
	rdks_200	0.0	0.6	2.2	0.0	0.2	1.6	0.0	0.2	1.6	
	rdks_300	0.0	0.9	2.7	0.0	0.5	2.0	0.0	0.5	2.0	18.5
	rdsrt_200	0.0	0.6	2.2	0.0	0.2	1.6	0.0	0.2	1.6	
	rdsrt_300	0.0	0.8	2.5	0.0	0.4	2.0	0.0	0.4	2.0	
enedis	kmeanspp_300	13.1	21.1	34.1	12.9	21.0	34.1	12.9	21.0	34.1	
	rdks_200	0.0	0.1	0.5	0.0	0.0	0.5	0.0	0.2	5.8	
	rdks_300	0.1	1.4	8.6	0.0	1.2	8.3	0.0	1.9	10.6	50.7
	rdsrt_200	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.4	7.8	
	rdsrt_300	0.1	1.1	7.8	0.0	1.0	7.8	0.0	1.2	10.2	
insurance	kmeanspp_300	0.1	13.5	26.8	0.0	13.3	26.4	0.0	13.2	26.4	
	rdks_200	0.1	0.8	3.9	0.0	0.6	3.8	0.0	0.7	3.7	
	rdks_300	0.0	2.3	10.7	0.0	2.2	10.6	0.0	2.2	10.6	37.1
	rdsrt_200	0.0	0.8	3.9	0.0	0.6	3.5	0.0	0.6	3.5	
	rdsrt_300	0.0	2.0	10.3	0.0	1.9	9.9	0.0	1.9	9.9	

Table 6.11: Deviation (%) to best-known solution constructed on instances generated by the query generation method, solved by the heuristics, matheuristic, and CPLEX, compared to generating the complete instance and applying h-KS.

6.4.3 Improving Query Generation

In this experiment, we select the three best pool generation methods and pair them with the best solver configuration for each database. We evaluate their performance compared to the baseline approach described in Chapter 4, where all queries are generated, and the complete TAP instance is solved. This generation takes under 10 minutes for the enedis database, an hour for the insurance database, and 6 hours for the flights database (we report the number of comparison queries in Table 6.1). As with previous experiments, a 1-hour time limit is imposed on the query generation-based approaches, including 10 minutes reserved for solving the TAP on the generated queries. The solutions are compared in terms of their objective function's deviation from the best-known solution.

The results of this experiment are shown in Table 6.11. Across all databases, the query generation strongly outperforms the baseline approach. We note that the kmeanspp_300, however, performs worse than the other query generation methods. This is likely due to its longer running time, which reduces the number of iterations of query generation that follow. Likewise, for rdks and rdsrt where smaller (200 queries) pools seem to perform better than larger (300 queries) pools. This is likely due to a larger number of queries yielding longer and, thus, fewer iterations of query generation. The significantly worse performance of the h-KS heuristic when solving a complete instance versus a partial instance produced by query generation may be

explained by its behavior. Indeed, with no possibility of discarding a query as long as it does not lead to violating an $\varepsilon-$constraint, it is possible that one or more high-interest (and low-cost) queries are added to the solution despite being detrimental to the solution in terms of distance. These detrimental queries might not be generated by our query generation method since they would be seen as not improving when solving the query generation problem (dual).

6.4.4 Running times

In this last series of experiments, we attempt to evaluate the minimum time needed by our approach to maintain a high-quality solution. For all databases, we try different running time budgets (up to an hour) with a reserved 10 minutes for running the matheuristic. We report the gap to the best-known solution in Figure 6.4.

As with the previous set of experiments, we see that initializing the query generation step using kmeanspp produces the worst results. We can confidently confirm that this is purely due to the running time of the approach as its performance approaches other methods as the time budget increases. We also notice that for the two smaller databases (flights and enedis), a 20-minute budget is sufficient, while a 30-minute budget for insurance can be used. Larger runtimes do not lead to computing significantly better solutions.

6.5 Conclusion

In this chapter, we designed, implemented, and tested a method to solve the TAP on an instance without generating it completely. We proposed several strategies to initialize our newly designed query generation method and identified two, namely rdks and rdsrt, that outperform the others. Our query generation method manages to construct better solutions than the h-KS heuristic when this one exploits the full set of queries. On larger databases, our method is also faster than generating this full set of queries.

Interestingly, it appears that the behavior of the MIP solver, regarding its parameters, varies widely from one database to the other. Whether this behavior can be reproduced with other solvers or predicted in any way remains out of the scope of this thesis but may be explored in future works. Likewise, the introduction of machine learning methods such as those used in our paper on lifelong pathways ([Chanson *et al.* 2021]) to learn interestingness could be explored as long as those models can be expressed as linear analytic expressions. Given that reinforcement learning based approaches were successfully used in EDA tasks, their use in our method could be explored, notably to replace the Dual-LP and the Dual-MIP models.

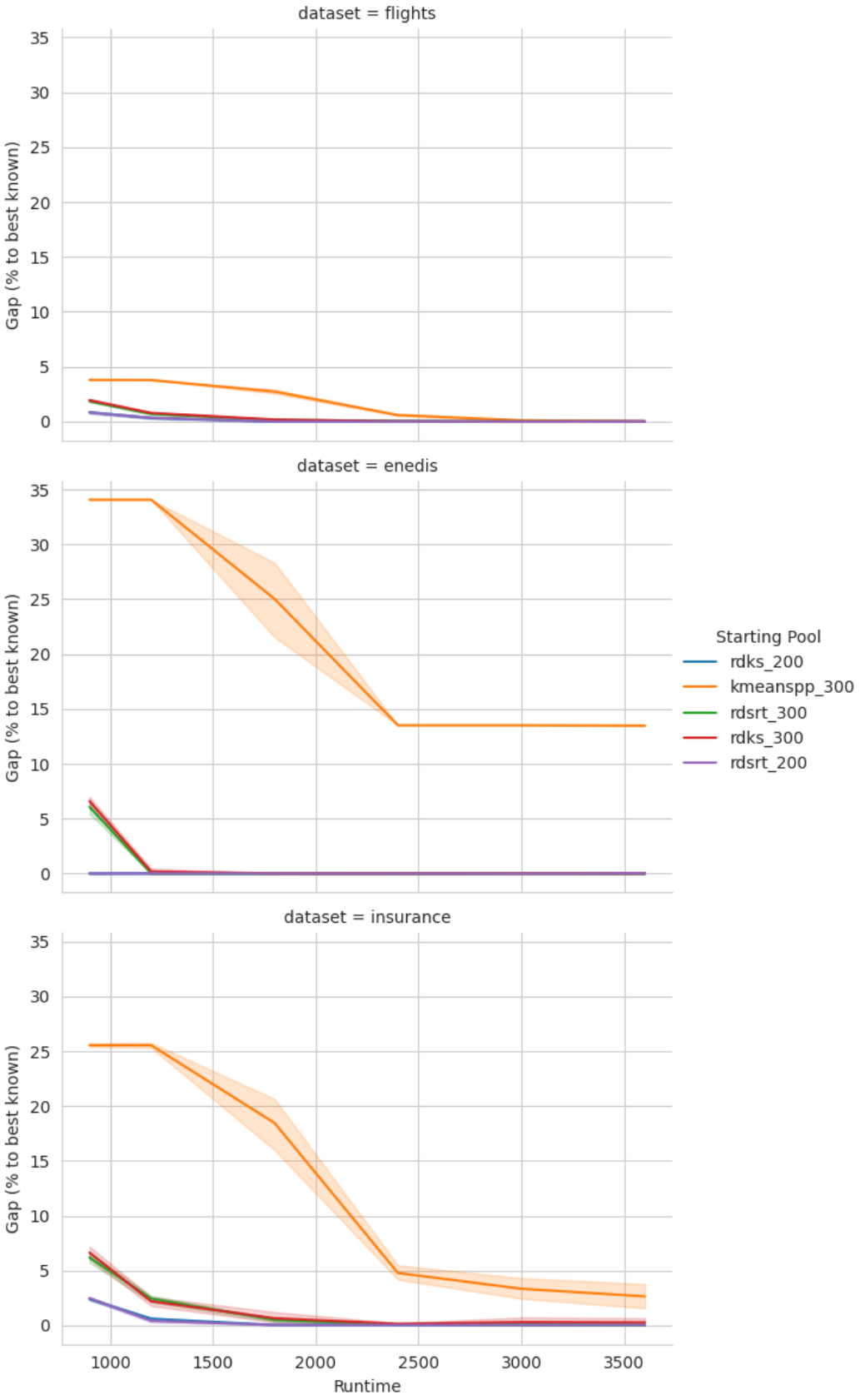

Figure 6.4: Deviation (%) to best-known solution constructed by the matheuristic using instances generated by the query generation method.

Chapter 7

Conclusion

We conclude this thesis by summarizing the various addressed challenges, raised issues, our contributions, and possible prospects following this work.

The main goal of the thesis is to explore the contributions that could be made to the field of automated exploratory data analysis (EDA) using tools and knowledge from the OR community. As discussed in Section 2.5, automated EDA problems can be modeled as extensions of the traveling salesperson problem that have been extensively surveyed and studied by the operations research community.

The first challenge of this thesis is to identify the types of insights and queries used by various authors in the relatively recent field of automated EDA. We elaborate on the comparison insights as studies have shown their prevalence among data workers' manual explorations. We formally define comparison insights and their associated comparison queries, thus providing a finite, countable search space for the traveling analyst problem.

Although previously defined in one of our publications ([Chanson *et al.* 2020]), we did not study its relation to transport problems and only provided a trivial heuristic. In this thesis, we establish the relationship between TAP and the orienteering problem and provide a mathematical formulation for the TAP. We propose two heuristics for solving TAP, focusing on different instance structures. We discuss possible cuts and instance pre-processing strategies to accelerate the solution process of the TAP when using a mathematical solver. We finally introduce two different matheuristic strategies to solve the TAP. They provide high-quality solutions to small-size instances faster than directly solving the MIP.

We develop a fully functional prototype constructing comparison query notebooks featuring comparison insights. To further this goal, we solve several practical challenges linked to the generation and solution of large TAP instances ($> 10^6$ queries). Notably, the execution of millions of non-parametric statistical tests. We conduct a user study, with its results indicating users are unable to distinguish optimal solutions from heuristic solutions.

Finally, drawing inspiration from column generation, we propose to avoid the costly construction of the complete TAP instances. We introduce the two-step query generation process that first constructs a pool of starting queries before iteratively finding improving queries. On larger databases, this process is faster than generating the complete TAP instances. It produces better quality solutions than our heuristic running on the complete set of queries.

Future works

In Chapter 5, we highlight the long time dedicated to constructing TAP instances in a real-world application. In Chapter 7, we propose an approach bypassing the need to generate the complete TAP instance. However, improvements can still be made to the instance construction methods proposed in Chapter 5. Notably, a large portion of the construction time can be directly attributed to the large number of non-parametric statistical tests executed. The use of non-parametric tests is mainly due to the absence of prior knowledge about the data. Making any distributional assumptions would be dubious in this context. In a separate work, we conducted preliminary experiments attempting to identify normally distributed samples in real-world open-source databases and apply parametric testing to them. These preliminary results are promising. By detecting additional common distributions and applying the relevant parametric tests, we believe this phase of the construction instance could be significantly accelerated. However, introducing further testing to choose parametric versus non-parametric tests should not introduce a large overhead. To avoid this overhead, a statistically sound rule should be designed. It should be able to identify the common distributions where parametric testing is applicable while falling back to non-parametric tests as a last resort.

A key aspect for automating EDA remains the quality of the used interestingness measure ([De Bie *et al.* 2022]). However, the goal of this thesis is not to contribute or improve an interestingness measure for EDA. We acknowledge the inevitable diversity of interestingness measures in the EDA community ([De Bie *et al.* 2022, Amer-Yahia *et al.* 2023b]) and put forward the TAP as a generic problem that adapts to many existing interestingness measures. Our heuristic (and matheuristic) solution approaches are fully compatible with this philosophy. They only require the pre-computed positive interestingness values for each query. On the other hand, the query generation method requires a linear analytical expression of the interest. However, this limitation could be overtaken. Notably, by taking inspiration from other work in the automated EDA field ([Bar El *et al.* 2019], a reinforcement learning (RL) method could replace the MIP to generate queries while retaining the general query generation algorithm. However, one of the main issues of current RL-based approaches is the long training time and poor reusability of the models. Fortunately, recent developments in RL ([Kobanda *et al.* 2023]) may allow us to learn efficiently even complex interestingness measures. Another direction could be to use machine learning to learn a linear (or easily linearizable) model that would act as a surrogate for the real interestingness measure within the dual-LP MIP. With simple models often associated with fast learning algorithms, it might even be possible to learn personalized interestingness measures ([Bie 2013]) for each user.

While testing our query generation algorithm (Chapter 7) we noticed that it outperforms the h-KS heuristic, solving the complete instance while only generating a few hundred queries. We speculate this is caused by high interestingness queries with relatively large distances from the others. An additional step removing such

queries from the solutions constructed by h-KS, could prove beneficial. Likewise, a periodic re-routing of the solution (using an off-the-shelf TSP solver) could be beneficial. However, it appears that some interestingness and distance measures for queries lead to very particular instances that require dedicated optimization algorithms. These instances may have only a few distinct interest or distance values. This may affect the behavior and the performance of the heuristics and the MIP solvers. Since this is dependent on the specific interest measure(s) and distance function(s) used in an application, it is hard to design a single heuristic that can offer good performance in all scenarios. It may be possible to use machine learning as a way to select an appropriate solution method based on instance characteristics.

Although we focus our efforts on developing a working prototype implementing the TAP for the comparison queries, there are several other types of insights, such as correlation or trend ([Zgraggen *et al.* 2018, Ding *et al.* 2019]). This is a major issue when envisioning a completely automated EDA system. Dealing with each type of insight would require not only more types of queries but potentially different interestingness measures for each insight type. This would obviously broaden the search space and thus produce larger instances of the TAP. But the main issue would be ensuring that interestingness measures are comparable even if computed by different means. [Ding *et al.* 2019] have attempted to solve this issue by using a p-value as part of their interestingness measure, as no matter the statistical test used, they always obtain a comparable value. However, as discussed by [Amer-Yahia *et al.* 2023b] and summarized in Table 2.1, statistical significance is only a single facet of interestingness.

Bibliography

[Agrawal & Srikant 2000] Rakesh Agrawal and Ramakrishnan Srikant. *Fast Algorithms for Mining Association Rules*. Proc. 20th Int. Conf. Very Large Data Bases VLDB, vol. 1215, 08 2000. (Cited on page 10.)

[Aligon *et al.* 2014] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi and Elisa Turricchia. *Similarity measures for OLAP sessions*. Knowledge and Information Systems, vol. 39, no. 2, pages 463–489, 2014. (Cited on pages 56 and 75.)

[Amer-Yahia & Roy 2018] Sihem Amer-Yahia and Senjuti Basu Roy. *Interactive Exploration of Composite Items*. In EDBT, pages 513–516, 2018. (Cited on page 12.)

[Amer-Yahia *et al.* 2023a] Sihem Amer-Yahia, Angela Bonifati, Lei Chen, Guoliang Li, Kyuseok Shim, Jianliang Xu and Xiaochun Yang. *From Large Language Models to Databases and Back: A discussion on research and education*, 2023. (Cited on page 12.)

[Amer-Yahia *et al.* 2023b] Sihem Amer-Yahia, Patrick Marcel and Verónika Peralta. *Data Narration for the People: Challenges and Opportunities*. In Julia Stoyanovich, Jens Teubner, Nikos Mamoulis, Evaggelia Pitoura, Jan Mühlig, Katja Hose, Sourav S. Bhowmick and Matteo Lissandrini, editors, Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, pages 855–858. OpenProceedings.org, 2023. (Cited on pages 1, 5, 7, 9, 10, 13, 100 and 101.)

[Archetti *et al.* 2015] Claudia Archetti, Ángel Corberán, Isaac Plana, José Maria Sanchis and M. Grazia Speranza. *A matheuristic for the Team Orienteering Arc Routing Problem*. European Journal of Operational Research, vol. 245, no. 2, pages 392–401, 2015. (Cited on page 21.)

[Arthur & Vassilvitskii 2007] David Arthur and Sergei Vassilvitskii. *k-means++: the advantages of careful seeding*. In ACM-SIAM Symposium on Discrete Algorithms, 2007. (Cited on page 78.)

[Bar El *et al.* 2019] Ori Bar El, Tova Milo and Amit Somech. *ATENA: An Autonomous System for Data Exploration Based on Deep Reinforcement Learning*. Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019. (Cited on pages 1, 8, 11, 72, 77 and 100.)

[Benjamini & Hochberg 1995] Yoav Benjamini and Yosef Hochberg. *Controlling The False Discovery Rate - A Practical And Powerful Approach To Multiple Testing*. J. Royal Statist. Soc., Series B, vol. 57, pages 289 – 300, 11 1995. (Cited on page 58.)

[Benjamini 2010] Yoav Benjamini. *Discovering the False Discovery Rate.* Journal of the Royal Statistical Society Series B: Statistical Methodology, vol. 72, no. 4, pages 405–416, 08 2010. (Cited on page 58.)

[Bianchessi *et al.* 2018] Nicola Bianchessi, Renata Mansini and M. Grazia Speranza. *A branch-and-cut algorithm for the Team Orienteering Problem.* International Transactions in Operational Research, vol. 25, no. 2, pages 627–635, 2018. (Cited on page 20.)

[Bie 2013] Tijl De Bie. *Subjective Interestingness in Exploratory Data Mining.* In Allan Tucker, Frank Höppner, Arno Siebes and Stephen Swift, editors, Advances in Intelligent Data Analysis XII - 12th International Symposium, IDA 2013, London, UK, October 17-19, 2013. Proceedings, volume 8207 of *Lecture Notes in Computer Science*, pages 19–31. Springer, 2013. (Cited on pages 8 and 100.)

[Blanton *et al.* 2017] Michael R. Blanton*et al. Sloan Digital Sky Survey IV: Mapping the Milky Way, Nearby Galaxies, and the Distant Universe.* The Astronomical Journal, vol. 154, no. 1, page 28, July 2017. (Cited on page 72.)

[Blount *et al.* 2020] 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. (Cited on pages 9 and 21.)

[Cao *et al.* 2012] Xin Cao, Lisi Chen, Gao Cong and Xiaokui Xiao. *Keyword-aware Optimal Route Search.* Proc. VLDB Endow., vol. 5, no. 11, pages 1136–1147, 2012. (Cited on page 12.)

[Chanson *et al.* 2020] Alexandre Chanson, Ben Crulis, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Stefano Rizzi and Panos Vassiliadis. *The Traveling Analyst Problem: Definition and Preliminary Study.* In Il-Yeol Song, Katja Hose and Oscar Romero, editors, Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT 2020 Joint Conference, DOLAP@EDBT/ICDT 2020, Copenhagen, Denmark, March 30, 2020, volume 2572 of *CEUR Workshop Proceedings*, pages 94–98. CEUR-WS.org, 2020. (Cited on pages 13, 25 and 99.)

[Chanson *et al.* 2021] Alexandre Chanson, Thomas Devogele, Nicolas Labroche, Patrick Marcel, Nicolas Ringuet and Vincent T'Kindt. *A Chain Composite Item Recommender for Lifelong Pathways.* In Matteo Golfarelli, Robert Wrembel, Gabriele Kotsis, A. Min Tjoa and Ismail Khalil, editors, Big Data Analytics and Knowledge Discovery, pages 55–66, Cham, 2021. Springer International Publishing. (Cited on pages 49, 50 and 96.)

[Chanson *et al.* 2022a] Alexandre Chanson, Nicolas Labroche, Patrick Marcel, Stefano Rizzi and Vincent T'kindt. *Automatic generation of comparison notebooks for interactive data exploration.* In Julia Stoyanovich, Jens Teubner, Paolo Guagliardo, Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish and Meihui Zhang, editors, Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022, pages 2:274–2:284. OpenProceedings.org, 2022. (Cited on pages 8, 36, 49, 74, 77 and 84.)

[Chanson *et al.* 2022b] Alexandre Chanson, Faten El Outa, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Willeme Verdeaux and Lucile Jacquemart. *Generating Personalized Data Narrations from EDA Notebooks.* In Kostas Stefanidis and Lukasz Golab, editors, Proceedings of the 24th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 25th International Conference on Extending Database Technology and the 25th International Conference on Database Theory (EDBT/ICDT 2022), Edinburgh, UK, March 29, 2022, volume 3130 of *CEUR Workshop Proceedings*, pages 91–95. CEUR-WS.org, 2022. (Cited on pages 8, 49 and 50.)

[Chao *et al.* 1996] I-Ming Chao, Bruce L. Golden and Edward A. Wasil. *A fast and effective heuristic for the orienteering problem.* European Journal of Operational Research, vol. 88, no. 3, pages 475–489, 1996. (Cited on page 20.)

[Chaudhuri *et al.* 1999] Surajit Chaudhuri, Eric Christensen, Goetz Graefe, Vivek R. Narasayya and Michael J. Zwilling. *Self-tuning technology in microsoft sql server.* IEEE Data Eng. Bull., vol. 22, no. 2, pages 20–26, 1999. (Cited on page 75.)

[Cunningham *et al.* 2004] 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. (Cited on page 22.)

[Dageville *et al.* 2004] Benoit Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zait and Mohamed Ziauddin. *Automatic sql tuning in oracle 10g.* In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 1098–1109, 2004. (Cited on page 75.)

[Das *et al.* 2019] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu and Surajit Chaudhuri. *Automatically Indexing Millions of Databases in Microsoft Azure SQL Database.* In Proceedings of the 2019 International Conference on Management of Data, SIGMOD '19, page 666–679, New York, NY, USA, 2019. Association for Computing Machinery. (Cited on page 75.)

[De Bie *et al.* 2022] Tijl De Bie, Luc De Raedt, José Hernández-Orallo, Holger H Hoos, Padhraic Smyth and Christopher KI Williams. *Automating Data Science: Prospects and Challenges*, 2022. (Cited on pages 10 and 100.)

[Della Croce *et al.* 2013] Federico Della Croce, Andrea Grosso and Fabio Salassa. *Matheuristics: Embedding MILP solvers into heuristic algorithms for combinatorial optimization problems.* Heuristics: Theory and Applications, pages 53–68, 02 2013. (Cited on pages 19 and 33.)

[Ding *et al.* 2019] Rui Ding, Shi Han, Yong Xu, Haidong Zhang and Dongmei Zhang. *QuickInsights: Quick and Automatic Discovery of Insights from Multi-Dimensional Data.* In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande and Tim Kraska, editors, Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, pages 317–332, 2019. (Cited on pages 1, 5, 7, 8, 9, 10, 12, 21, 74 and 101.)

[El *et al.* 2020] Ori Bar El, Tova Milo and Amit Somech. *Automatically Generating Data Exploration Sessions Using Deep Reinforcement Learning.* In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini and Hung Q. Ngo, editors, Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, pages 1527–1537, 2020. (Cited on pages 6, 51, 55, 56 and 68.)

[Engquist 1982] Michael Engquist. *A Successive Shortest Path Algorithm for The Assignment Problem.* INFOR: Information Systems and Operational Research, vol. 20, no. 4, pages 370–384, 1982. (Cited on page 29.)

[Feillet *et al.* 2005] Dominique Feillet, Pierre Dejax and Michel Gendreau. *Traveling Salesman Problems with Profits.* Transportation Science, vol. 39, no. 2, pages 188–205, 2005. (Cited on page 14.)

[Feillet 2010] Dominique Feillet. *A tutorial on column generation and branch-and-price for vehicle routing problems.* 4OR, vol. 8, pages 407–424, 12 2010. (Cited on page 71.)

[Fischetti & Fischetti 2018] Martina Fischetti and Matteo Fischetti. Matheuristics, pages 121–153. Springer International Publishing, Cham, 2018. (Cited on pages 18 and 19.)

[Fischetti *et al.* 1998] Matteo Fischetti, Juan José Salazar González and Paolo Toth. *Solving the Orienteering Problem through Branch-and-Cut.* INFORMS Journal on Computing, vol. 10, no. 2, pages 133–148, 1998. (Cited on page 20.)

[Francia *et al.* 2021] 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. (Cited on pages 9 and 22.)

[Francia *et al.* 2022] Matteo Francia, Patrick Marcel, Verónika Peralta and Stefano Rizzi. *Enhancing Cubes with Models to Describe Multidimensional Data*. Inf. Syst. Frontiers, vol. 24, no. 1, pages 31–48, 2022. (Cited on page 8.)

[Garcia-Molina *et al.* 2002] Garcia-Molina, Hector Hector, Ullman, Jeffrey D, Widom and Jennifer. Database systems: The complete book. 01 2002. (Cited on page 22.)

[Garcia-Molina *et al.* 2009] H. Garcia-Molina, J.D. Ullman and J. Widom. Database systems: The complete book. Pearson international edition. Pearson Prentice Hall, 2009. (Cited on page 75.)

[Garey & Johnson 1990] Michael R. Garey and David S. Johnson. Computers and intractability; a guide to the theory of np-completeness. W. H. Freeman and Co., USA, 1990. (Cited on page 15.)

[Geng & Hamilton 2006] Liqiang Geng and Howard J. Hamilton. *Interestingness Measures for Data Mining: A Survey*. ACM Comput. Surv., vol. 38, no. 3, page 9–es, sep 2006. (Cited on pages 6 and 75.)

[Giacometti *et al.* 2009] Arnaud Giacometti, Patrick Marcel, Elsa Negre and Arnaud Soulet. *Query recommendations for OLAP discovery driven analysis*. In International Workshop on Data Warehousing and OLAP, 2009. (Cited on page 10.)

[Gionis *et al.* 2014] Aristides Gionis, Theodoros Lappas, Konstantinos Pelechrinis and Evimaria Terzi. *Customized tour recommendations in urban areas*. In WSDM, pages 313–322, 2014. (Cited on page 12.)

[Gunawan *et al.* 2016] Aldy Gunawan, Hoong Chuin Lau and Pieter Vansteenwegen. *Orienteering Problem: A survey of recent variants, solution approaches and applications*. European Journal of Operational Research, vol. 255, no. 2, pages 315–332, December 2016. (Cited on page 19.)

[Gurobi 2016] Gurobi. *Algorithms in Gurobi*, 2016. (Cited on page 17.)

[Hamming 1950] R. W. Hamming. *Error detecting and error correcting codes*. The Bell System Technical Journal, vol. 29, no. 2, pages 147–160, 1950. (Cited on pages 19 and 33.)

[Helsgaun 2000] Keld Helsgaun. *An effective implementation of the Lin–Kernighan traveling salesman heuristic*. European Journal of Operational Research, vol. 126, no. 1, pages 106–130, 2000. (Cited on page 31.)

[Hu & Lim 2014] Qian Hu and Andrew Lim. *An iterative three-component heuristic for the team orienteering problem with time windows.* European Journal of Operational Research, vol. 232, no. 2, pages 276–286, 2014. (Cited on page 20.)

[IBM 2021] IBM. *IBM CPLEX Documentation*, November 2021. (Cited on pages 16, 17 and 91.)

[Idreos *et al.* 2015] Stratos Idreos, Olga Papaemmanouil and Surajit Chaudhuri. *Overview of Data Exploration Techniques.* In Proceedings of SIGMOD, pages 277–281, 2015. (Cited on pages 1, 5 and 10.)

[Kahng & Reda 2004] Andrew B. Kahng and Sherief Reda. *Match twice and stitch: a new TSP tour construction heuristic.* Operations Research Letters, vol. 32, no. 6, pages 499–509, 2004. (Cited on page 29.)

[Kara *et al.* 2016] Imdat Kara, Papatya Sevgin Bicakci and Tusan Derya. *New Formulations for the Orienteering Problem.* Procedia Economics and Finance, vol. 39, pages 849–854, 2016. (Cited on page 20.)

[Kellerer *et al.* 2004] Hans Kellerer, Ulrich Pferschy and David Pisinger. Knapsack problems. 01 2004. (Cited on page 15.)

[Kirby 2003] Maurice Kirby. Operational research in war and peace: The british experience from the 1930s to 1970. 06 2003. (Cited on page 14.)

[Kobanda *et al.* 2023] Anthony Kobanda, Valliappan C. A., Joshua Romoff and Ludovic Denoyer. *Learning Computational Efficient Bots with Costly Features.* In Proceedings of IEEE CoG, 2023. (Cited on page 100.)

[Lee & Lee 2018] S. Lee and D. K. Lee. *What is the proper way to apply the multiple comparison test?* Korean J Anesthesiol, vol. 71, no. 5, pages 353–360, Oct 2018. (Cited on page 10.)

[Ma *et al.* 2021] Pingchuan Ma, Rui Ding, Shi Han and Dongmei Zhang. *MetaInsight: Automatic Discovery of Structured Knowledge for Exploratory Data Analysis.* In Guoliang Li, Zhanhuai Li, Stratos Idreos and Divesh Srivastava, editors, SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, pages 1262–1274, 2021. (Cited on pages 6, 7, 8, 10, 12 and 60.)

[Ma *et al.* 2023a] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han and Dongmei Zhang. *Demonstration of InsightPilot: An LLM-Empowered Automated Data Exploration System.* CoRR, vol. abs/2304.00477, 2023. (Cited on page 12.)

[Ma *et al.* 2023b] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han and Dongmei Zhang. *XInsight: EXplainable Data Analysis Through The Lens of Causality.* Proc. ACM Manag. Data, vol. 1, no. 2, jun 2023. (Cited on pages 5 and 6.)

[Marcel *et al.* 2019] Patrick Marcel, Verónika Peralta and Panos Vassiliadis. *A Framework for Learning Cell Interestingness from Cube Explorations.* In Proceedings of ADBIS, pages 425–440, 2019. (Cited on pages 6 and 56.)

[Miller *et al.* 1960] C. E. Miller, A. W. Tucker and R. A. Zemlin. *Integer Programming Formulation of Traveling Salesman Problems.* Journal of the ACM, vol. 7, no. 4, page 326–329, 1960. (Cited on page 27.)

[Milo & Somech 2018] Tova Milo and Amit Somech. *Next-Step Suggestions for Modern Interactive Data Analysis Platforms.* In SIGKDD, pages 576–585. ACM, 2018. (Cited on page 6.)

[Milo & Somech 2020] Tova Milo and Amit Somech. *Automating Exploratory Data Analysis via Machine Learning: An Overview.* In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, page 2617–2622, New York, NY, USA, 2020. Association for Computing Machinery. (Cited on page 9.)

[Personnaz *et al.* 2021] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Équille, Maximilian Fabricius and Srividya Subramanian. *DORA THE EXPLORER: Exploring Very Large Data With Interactive Deep Reinforcement Learning.* In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang and Hanghang Tong, editors, CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021, pages 4769–4773, 2021. (Cited on pages 8 and 11.)

[Personnaz *et al.* 2022] Aurélien Personnaz, Brit Youngmann and Sihem Amer-Yahia. *EDA4SUM: Guided Exploration of Data Summaries.* Proc. VLDB Endow., vol. 15, no. 12, page 3590–3593, aug 2022. (Cited on pages 8 and 72.)

[Pisinger 1995] David Pisinger. *An expanding-core algorithm for the exact 0–1 knapsack problem.* European Journal of Operational Research, vol. 87, no. 1, pages 175–187, November 1995. (Cited on page 16.)

[Pisinger 2005] David Pisinger. *Where are the hard knapsack problems?* Computers & Operations Research, vol. 32, no. 9, pages 2271–2284, 2005. (Cited on page 36.)

[Razmadze *et al.* 2022] Kathy Razmadze, Yael Amsterdamer, Amit Somech, Susan B. Davidson and Tova Milo. *SubTab: Data Exploration with Informative Sub-Tables.* In Proceedings of the 2022 International Conference on Management of Data, pages 2369–2372. ACM, 2022. (Cited on pages 8 and 9.)

[Rojas *et al.* 2017] 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. (Cited on page 9.)

[Roy *et al.* 2011] Senjuti Basu Roy, Gautam Das, Sihem Amer-Yahia and Cong Yu. *Interactive itinerary planning.* In ICDE, pages 15–26, 2011. (Cited on pages 12 and 21.)

[Salton & Buckley 1988] Gerard Salton and Christopher Buckley. *Term-weighting approaches in automatic text retrieval.* Information Processing & Management, vol. 24, no. 5, pages 513–523, 1988. (Cited on page 50.)

[Sarawagi *et al.* 1998] Sunita Sarawagi, Rakesh Agrawal and Nimrod Megiddo. *Discovery-Driven Exploration of OLAP Data Cubes.* In Proceedings of EDBT, pages 168–182, 1998. (Cited on page 6.)

[Sarawagi 1999] Sunita Sarawagi. *Explaining Differences in Multidimensional Aggregates.* In Proceedings of VLDB, pages 42–53, 1999. (Cited on page 6.)

[Sarawagi 2000] Sunita Sarawagi. *User-Adaptive Exploration of Multidimensional Data.* In Proceedings of VLDB, pages 307–316, 2000. (Cited on page 6.)

[Sathe & Sarawagi 2001] Gayatri Sathe and Sunita Sarawagi. *Intelligent Rollups in Multidimensional OLAP Data.* In Proceedings of VLDB, pages 531–540, 2001. (Cited on page 6.)

[Sawtell-Rickson 2023] Jye Sawtell-Rickson. *My First Exploratory Data Analysis with ChatGPT*, 2023. (Cited on page 12.)

[Schrijver 1986] Alexander Schrijver. Theory of linear and integer programming. 1986. (Cited on pages 16 and 17.)

[Shi *et al.* 2020] Danqing Shi, Xinyue Xu, Fuling Sun, Yang Shi and Nan Cao. *Calliope: Automatic Visual Data Story Generation from a Spreadsheet.* IEEE Transactions on Visualization and Computer Graphics, vol. 27, pages 453–463, 2020. (Cited on pages 7, 8 and 11.)

[Siddiqui *et al.* 2021] Tarique Siddiqui, Surajit Chaudhuri and Vivek R. Narasayya. *COMPARE: Accelerating Groupwise Comparison in Relational Databases for Data Analytics.* Proceedings of VLDB Endow., vol. 14, no. 11, pages 2419–2431, 2021. (Cited on pages 9 and 21.)

[Singhal & Google 2001] Amit Singhal and I. Google. *Modern Information Retrieval: A Brief Overview.* IEEE Data Engineering Bulletin, vol. 24, 01 2001. (Cited on page 50.)

[Tang *et al.* 2017] 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. (Cited on pages 5, 6, 7, 8, 9, 10, 12, 21, 51, 52 and 56.)

[T'kindt & Billaut 2006] Vincent T'kindt and Jean-Charles Billaut. Multicriteria scheduling - theory, models and algorithms (2. ed.). Springer, 2006. (Cited on page 26.)

[T'Kindt 2023] Vincent T'Kindt. *The Marriage of Matheuristics and Scheduling.* In Proceedings of Scheduling seminar, 2023. (Cited on pages 18 and 19.)

[Tsiligirides 1984] Theodore Tsiligirides. *Heuristic Methods Applied to Orienteering.* Journal of the Operational Research Society, vol. 35, no. 9, pages 797–809, 1984. (Cited on pages 19 and 20.)

[Tukey 1977] John W. Tukey. Exploratory data analysis. Addison-Wesley, 1977. (Cited on pages 1 and 5.)

[Vansteenwegen *et al.* 2011] Pieter Vansteenwegen, Wouter Souffriau and Dirk Van Oudheusden. *The orienteering problem: A survey.* European Journal of Operational Research, vol. 209, no. 1, pages 1–10, February 2011. (Cited on page 19.)

[Wang *et al.* 2020] Y. Wang, Z. Sun, H. Zhang, W. Cui, K. Xu, X. Ma and D. Zhang. *DataShot: Automatic Generation of Fact Sheets from Tabular Data.* IEEE Trans Vis Comput Graph, vol. 26, no. 1, pages 895–905, Jan 2020. (Cited on pages 6, 7 and 11.)

[Winston 2022] Wayne L Winston. Operations research: applications and algorithms, chapter 4, pages 197–201. Cengage Learning, 2022. (Cited on page 72.)

[Wren & Holliday 1972] Anthony Wren and Alan Holliday. *Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points.* Operational Research Quarterly (1970-1977), vol. 23, no. 3, pages 333–344, 1972. (Cited on page 20.)

[Xing *et al.* 2002] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart J. Russell. *Distance Metric Learning with Application to Clustering with Side-Information.* In NIPS, pages 505–512, 2002. (Cited on page 50.)

[Young 2016] Neal E. Young. *Greedy Set-Cover Algorithms.* In Encyclopedia of Algorithms, pages 886–889. Springer, 2016. (Cited on page 60.)

[Yu *et al.* 2019] Qinxiao Yu, Kan Fang, Ning Zhu and Shoufeng Ma. *A matheuristic approach to the orienteering problem with service time dependent profits.* European Journal of Operational Research, vol. 273, no. 2, pages 488–503, 2019. (Cited on page 21.)

[Zgraggen *et al.* 2018] 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. (Cited on pages 1, 5, 6, 7, 9, 10, 21, 51, 52, 55, 56, 58 and 101.)

[Zhao *et al.* 2023] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie and Ji-Rong Wen. *A Survey of Large Language Models*, 2023. (Cited on page 12.)

Alexandre CHANSON

Optimisation et analyse interactive de données : le Problème du Voyageur de Données

Cette thèse contribue à l'automatisation de l'AED (analyse exploratoire des données). L'AED est un processus itératif qui consiste à analyser des données en effectuant des actions (telle qu'une requête sur des données), à recevoir le résultat et à décider de l'étape suivante. L'objectif final de l'AED est l'extraction de trouvailles, des fragments d'informations utiles étayées par les données. L'automatisation de l'AED nécessite de surmonter plusieurs obstacles, dont notamment l'identification et la représentation des informations les plus intéressantes présentent dans une base de données. Dans cette thèse, nous abordons le problème de la construction d'une séquence de requêtes représentant des informations pertinentes. Nous introduisons et étudions le problème d'optimisation associé, nommé le problème du voyageur de données (TAP). Nous établissons la relation entre le TAP et une famille de problèmes de transport classiques appelés problèmes d'orientation. En étudiant la littérature, nous avonsidentifié l'action de comparaison de données comme la principale activité des travailleurs de la donnée. Nous définissons formellement les trouvailles de comparaison et les requêtes de comparaison associées. Nous proposons différentes stratégies de résolution selon les tailles des bases de données, dont notamment des matheuristiques. Celles-ci tirent profit des solveurs mathématiques pour construire des solutions de haute qualité au TAP, notamment pour les bases de données de petite taille. Pour les bases de données de grandes taille, nous nous sommes d'abord chargé de la construction des instances du TAP. Une tache complexe, nécessitant la mise en œuvre de plusieurs stratégies d'optimisation notamment pour exécuter des millions de tests statistiques rapidement. Ces instances peuvent ensuite être résolu grâce a deux heurisitques. Finallement, pour eviter ce process de construction des instances, nous nous inspirons du processus de génération de colonnes. Nous introduisons la génération de requêtes, une méthode capable de résoudre le TAP sans explorer l'ensemble des requêtes de comparaison associées à une base de données.

Mots clés : Bases de données, problème de transport sous contraintes, analyse exploratoire de données, requêtes de comparaisons.

This thesis contributes to the automation of EDA (exploratory data analysis). EDA is the iterative process of performing an action (such as a query on data), receiving the result, and deciding on the next step. The final goal of EDA is the extraction of insights, fragments of useful information justified by the data. Automating EDA requires overcoming several hurdles, notably identifying and correctly presenting the most interesting insights from a database. In this thesis, we address the problem of constructing a sequence of queries representing relevant insights. We introduce and study the associated optimization problem, called the traveling analyst problem (TAP). We establish the relation between TAP and a family of classic transport problems named orienteering problems. Surveying the literature, we identify the comparison task as the main staple of data workers. We formally define the associated comparison insights and comparison queries. We propose different solution strategies for small and large databases. They include matheuristics, a recent development in the operation research community. Matheuristics take advantage of mathematical solvers in their construction. They proved capable of producing high-quality solutions to the TAP for small-size databases. For large-size databases, we first tackled the challenging construction of the TAP instances, requiring several optimization strategies to execute millions of statistical tests in less than an hour. Finally, we take inspiration from the process of column generation. We introduce query generation, a method that is capable of solving the TAP without exploring the full set of comparison queries associated with a database.

Keywords: Databases, Orienteering, Exploratory data analysis, comparison queries.