
Master's Thesis

# A Multi-Start Simulated Annealing Strategy for Data Lake Organization Problem

Danilo Fernandes Costa
dfc@ic.ufal.br

Advisers:

Dr. André Luiz Lins de Aquino
Dr. Rian Gabriel Santos Pinheiro

Maceió
March 26, 2024

Danilo Fernandes Costa

# A Multi-Start Simulated Annealing Strategy for Data Lake Organization Problem

A thesis submitted by Danilo Fernandes Costa in partial fulfillment of the requirements for the degree of Master of Science in Informatics at the Federal University of Alagoas, Computing Institute.

Advisers:

Dr. André Luiz Lins de Aquino

Dr. Rian Gabriel Santos Pinheiro

Maceió
March 26, 2024

**Catalogação na fonte**
**Universidade Federal de Alagoas**
**Biblioteca Central**
**Divisão de Tratamento Técnico**
Bibliotecária: Helena Cristina Pimentel do Vale – CRB4 –661

C837m Costa, Danilo Fernandes.
A multi-start simulated annealing strategy for data lake organization problem / Danilo Fernandes Costa. - 2024.
54 f : il.

Orientadores: André Luiz Lins de Aquino e Rian Gabriel Santos Pinheiro.
Dissertação (mestrado em Ciência da Informática) – Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2024.

Bibliografia: f. 47-54.

1. Data lake. 2. Organização. 3. Meta-heurística. I. Título.

CDU: 004.6

**MINISTÉRIO DA EDUCAÇÃO**
UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
Av. Lourival Melo Mota, S/N, Tabuleiro do Martins, Maceió - AL, 57.072-970
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (PROPEP)
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

## Folha de Aprovação

DANILO FERNANDES COSTA

# UMA ESTRATÉGIA MULTI-START SIMULATED ANNEALING PARA O PROBLEMA DA ORGANIZAÇÃO DE DADOS EM DATA LAKE

# A MULTI-START SIMULATED ANNEALING STRATEGY FOR DATA LAKE ORGANIZATION PROBLEM

Dissertação submetida ao corpo docente do Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas e aprovada em 19 de abril de 2024.

**Banca Examinadora:**

ANDRE LUIZ LINS DE AQUINO:03235015400
Assinado de forma digital por ANDRE LUIZ LINS DE AQUINO:03235015400
Dados: 2024.05.14 10:07:26 -03'00'

**Prof. Dr. ANDRE LUIZ LINS DE AQUINO**
UFAL – Instituto de Computação
**Orientador**

gov.br Documento assinado digitalmente
RIAN GABRIEL SANTOS PINHEIRO
Data: 15/05/2024 11:58:01-0300
Verifique em https://validar.iti.gov.br

**Prof. Dr. RIAN GABRIEL SANTOS PINHEIRO**
UFAL – Instituto de Computação
**Coorientador**

gov.br Documento assinado digitalmente
FABIANE DA SILVA QUEIROZ
Data: 15/05/2024 11:46:18-0300
Verifique em https://validar.iti.gov.br

**Profa. Dra. FABIANE DA SILVA QUEIROZ**
UFAL – Campus de Engenharia e Ciências Agrárias
**Examinadora Interna**

gov.br Documento assinado digitalmente
FABIO JOSE COUTINHO DA SILVA
Data: 14/05/2024 23:27:10-0300
Verifique em https://validar.iti.gov.br

**Prof. Dr. FABIO JOSE COUTINHO DA SILVA**
UFAL – Instituto de Computação
**Examinador Interno**

gov.br Documento assinado digitalmente
IGOR MACHADO COELHO
Data: 14/05/2024 20:09:40-0300
Verifique em https://validar.iti.gov.br

**Prof. Dr. IGOR MACHADO COELHO**
UFF - Universidade Federal Fluminense
**Examinador Externo**

# Resumo

*Data Lake* é a solução para *Big Data* que mais tem recebido atenção na atualidade. Sua principal característica é a capacidade de administrar imensos volumes de dados heterogêneos em seu formato bruto. Contudo, isto torna o acesso, a gestão e a exploração de dados mais complexos. Esse desafio define um problema organizacional. O Problema de Organização de *Data Lake* consiste na geração de uma estrutura navegacional otimizada para reduzir o esforço do usuário na exploração de todos os dados disponíveis. O objetivo é encontrar uma organização de dados que maximize a probabilidade esperada de descoberta de tabelas durante a navegação do utilizador. Para este problema, propomos uma metaheurística de recozimento simulado e comparamo-la com a solução *Organize* da literatura em instâncias de referência. Propomos também uma variação mais eficiente que elimina os cálculos excessivos. As instâncias são amostras do *Socrata Open Data Lake* com tópicos variados e dados abertos de entidades governamentais de todo o mundo. Para validar as nossas propostas, realizamos uma análise estatística utilizando um teste não paramétrico, que confirmou o domínio da nossa proposta sobre o estado-da-arte. Nossa melhor proposta foi mais eficiente e aumentou a probabilidade esperada de descoberta de tabelas em até $44\%$. Assim, nossa estratégia pode encontrar melhores soluções nos *benchmarks* avaliados mesmo sem analisá-los exaustivamente e explorar mais efetivamente o espaço de soluções.

**Keywords**: *Data Lake*, Descoberta de datasets, Taxonomia, Otimização.

# Abstract

Data Lake is the solution for Big Data that has received the most attention recently. Its main feature is handling vast volumes of heterogeneous data in its raw format. However, this makes data access, management, and exploration more complex. Such a challenge defines the organizational problem. The Data Lake Organization Problem comprises optimized data navigation structures generation to reduce the user's time exploring all available data. The goal is to find a data organization that maximizes the expected probability of table discovery during user navigation. For this problem, we propose a simulated annealing metaheuristic and compare it with the *Organize* literature solution on benchmark instances. We also propose a more efficient variation that prunes excessive computations. The instances are Socrata Open Data Lake samples with varying topics and open data from government entities worldwide. To validate our proposals, we performed statistical analysis using a non-parametric test, which confirmed the dominance of our proposition over the state-of-the-art. Our best proposal was more efficient and increased the expected probability of table discovery up to $44\%$. Thus, our strategy can find better solutions in the benchmarks evaluated even without exhaustively analyzing all of them and more effectively exploring the space of solutions.

**Keywords**: Data Lake, Dataset Discovery, Taxonomy, Optimization.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1

# Introduction

## 1.1 Motivation

Big data has become one of the most critical databases and distributed systems research challenges. Unprecedented volume, variety, and high velocity of data must be collected, stored, and processed to ensure integrity and generate value. The International Data Corporation forecasts that the global amount of data will reach 175 Zettabytes in 2025, ten times the volume of data generated by 2016 (Reinsel et al., 2018). According to this report, Internet of Things (IoT) devices will produce most of this data.

Efficient analytical systems are desirable and commonly indispensable for IoT and Big Data (Fernandes et al., 2023a). There are effective systems to store and analyze ordinary IoT data, but they are limited to specific data types (e.g., trajectory data) (Bakli et al., 2019; Solmaz et al., 2019). The systems are helpful for many applications since we can deploy IoT devices for specific tasks (e.g., traffic monitoring). However, IoT data analysis is not only required for applications with a single type of device deployed by a single organization. Applications with varying devices provided by different organizations are just as essential (Sasaki, 2022). Furthermore, an integration scenario of heterogeneous devices is the expected setting for smart city environments (Hashem et al., 2016).

In smart cities, we collect big data from several heterogeneous devices and systems, store, integrate, and analyze holistically (Ramos et al., 2023). For example, the data can refer to the geospatial indication of autonomous cars and drones' location over time; text and images from social media; video from surveillance systems; temperature, humidity, and pollution measurements collected by sensors distributed over the environment; and conventional structured data from multiple information systems (Talebkhah et al., 2021).

Data Lakes have been receiving attention due to their capacity to handle general and specific applications independently of big data variety. Traditional, time-tested solutions used data warehousing for management and processing. A data warehouse is a classic integrated storage

system designed to perform massive analysis (Stefanowski et al., 2017). However, while still relevant and robust for structured known sources, these solutions are not suited for semi-structured and unstructured data (Zagan and Danubianu, 2020).

Data lakes are appropriate solutions to scalable systems, enabling storage and analysis of heterogeneous data in its raw state for knowledge extraction (Fernandes et al., 2023a). This approach requires metadata generation and management to provide a high-quality data catalog through well-defined policies and tools. In addition, users have access to integration features, logical and physical organization, and scalable storage and processing power (Sawadogo and Darmont, 2021). These features make Data Lake an effective solution for data management and value generation for smart cities.

Considering a complex and heterogeneous system like a data lake, a robust metadata management system is imperative for optimized value extraction and insights (Francia et al., 2021). One of the functionalities of this system is dataset discovery. We typically formulate the dataset discovery as a search problem. In one version of the problem, the query is a set of keywords, and the goal is to find tables relevant to the keywords (Brickley et al., 2019). Alternatively, the query can be a table, and the problem is to find appropriate, joinable, or unionable tables (Zhu et al., 2016; Bharadwaj et al., 2021; Nargesian et al., 2018).

A complementary alternative to search is navigation. A user navigates through an organizational structure to find tables of interest in this paradigm. Such a mechanism allows expert Data Lake users to browse available topics, going from more general concepts to specific concepts (Nargesian et al., 2020). This approach saves time while exploring the available data and in vaguely defined searches since it is unnecessary to check everything stored.

The first effort to directly address this challenge was in the proposal of the Data Lake Organization Problem (DLOP) (Nargesian et al., 2020). In this, user browsing is modeled as a probabilistic graph whose optimization leads to a data organization that maximizes the expected probability of discovering tables during user navigation. Although there are some proposals for generating metadata graphs to integrate data in data lakes (Fernandez et al., 2018b; Aso et al., 2020; Yu et al., 2021a), these are not suitable for navigating (Nargesian et al., 2023). To the best of our knowledge, all the studies dealing with DLOP use the same heuristic to optimize it, namely the *'Organize'* algorithm (Nargesian et al., 2020, 2023; Ouellette et al., 2021).

However, organizing data into navigable structures remains a difficult and time-consuming task due to the data lake volume and variety. Given its relevance and complexity, we present a new Data Lake Organization Problem strategy. We propose a multi-start simulated annealing metaheuristic, called here just SA, which explores two neighborhood structures: add edge and delete node. Although suggested in the literature, these operations modify the probabilistic graph and have not been described algorithmically. Their foundation and complexity lie in guaranteeing the consistency of all probabilities after a modification, which requires coherent graph traversal algorithms based on dynamic programming. For these, we show how to implement them efficiently. We also propose a more efficient variation of SA that prunes excessive computations,

called Fast SA.

In addition to the existing efforts in addressing the Data Lake Organization Problem (DLOP), our research introduces a novel approach that aims to overcome the limitations of current methodologies. While previous studies have predominantly relied on the 'Organize' algorithm as a heuristic for optimizing data organization within data lakes, our proposed multi-start simulated annealing metaheuristic represents a significant departure from traditional methods. By exploring two distinct neighborhood structures - namely, the addition of edges and the deletion of nodes - our approach offers a more comprehensive and flexible framework for optimizing the probabilistic graph that models user browsing behavior.

Importantly, our methodology addresses a key gap in the literature by algorithmically describing these operations and ensuring the consistency of probabilities throughout the graph. Furthermore, we demonstrate the efficiency of our implementation, highlighting its potential to streamline the process of organizing data within data lakes. This novel strategy contributes to advancing the field of data lake management. It offers practical benefits for organizations that manage large volumes and diverse data types.

The main reason behind selecting SA for our study is rooted in its capacity to navigate complex, non-convex search spaces (Delahaye et al., 2018) efficiently. Unlike other trajectory metaheuristics such as GRASP, ILS, and Tabu Search, SA does not need to scan the entire neighborhood structure to select the next potential solution. This quality becomes especially crucial when dealing with a large neighborhood and confronting high time and space complexity during fitness evaluations such as those arising from big data problems (Ceschia and Schaerf, 2024).

The SA's ability to escape local optima, coupled with its flexibility in handling diverse problem structures, aligns well with the intricacies of our optimization problem (Heu, 2010). Furthermore, its simplicity of implementation is advantageous, considering the high computational demands associated with fitness evaluations in our context. SA's probabilistic acceptance of inferior solutions provides a valuable trade-off between exploration and exploitation in scenarios where fitness evaluations are computationally expensive. It is particularly well-suited for our problem domain (Delahaye et al., 2018).

We introduce the first set of benchmark data lakes for DLOP, leveraging those to compare our metaheuristic with the state-of-the-art one. We populate these instances with open data involving various topics, varying in context and size. A fair comparison between the metaheuristics was possible by tuning their hyper-parameters. We evaluate in terms of effectiveness, efficiency, and success probability.

First, we perform a non-parametric statistical test on solution fitness obtained in repeated experiments. In the second, we investigate the time to reach a reasonable solution. In the latter, we evaluate the navigation quality by simulating a user's exploration. For benchmarking, we consider a set of 30 data lakes sampled from the Socrata (Socrata, 2022), an open data lake populated with data from government entities from different countries. These data lake instances vary in size and context and contain data related to finances, social services, demographics,

infrastructure, health, environment, public safety, education, economy, and transportation.

In the effectiveness analysis, our proposals excelled in all instances, showing an improvement in the solution quality by up to $44\%$. For a better comparison, we ran the Mann-Whitney test (Mann and Whitney, 1947) on the effectiveness of solutions. We obtained p-values lower than $10^{-2}$, which shows the dominance of our approaches over the competition. In the efficiency analysis, we found reasonable solutions for an instance $75\times$ faster than the literature in most trials. Finally, we have achieved organizations with navigability up to $5\%$ better than in the literature. All analyses revealed that our proposals increases the expected probability of discovering tables and demands less time to find a suitable solution compared to the best literature approach.

## 1.2 Related Work

Taxonomies and semantic graphs are strategies often applied to enrich the data lake metadata that can provide hierarchical navigation (Gupta et al., 2017; Futia et al., 2020). Such structures are imported from existing and curated ones or automatically induced from data (Torregrossa et al., 2021; Yu et al., 2021a). Those allow annotating data with interrelated entities having different levels of granularity. Although they enable browsing through entities and faceted-search over them, each data source may be associated with various entities (Aso et al., 2020). Furthermore, these approaches are designed for something other than practical navigation, providing a hard browsing (Nargesian et al., 2023). Then, an organizational structure optimized to support navigation and exploration over tables in data lakes is advisable (Nargesian et al., 2020).

Yu et al. (2021a) developed an automated tool to build them for metadata representation. Their approach recognizes entities, concepts, properties, and relations by extracting statistical features and similarities from disparate data sources. The objective is to semantically integrate all data under a unified metadata model enriched with knowledge graphs. A similar metadata model is the Enterprise Knowledge Graph (EKG) proposed by Fernandez et al. (2018b). EKGs are built human effortlessly by `Aurum`, a metadata system that captures relationships between data columns and references from knowledge graphs. This system offers a significant advantage over the former by addressing data discovery through a search engine on top of the EKG. Yet, such approaches provide lazy and low successful browsing (Nargesian et al., 2023).

Nargesian et al. (2020, 2023) proposed a suitable framework. They have modeled the data lake organization as a graph, where each node represents a data topic. Browsing through such topics, the user can obtain relevant data sources when a keyword query needs to be better defined. Further, they define the DLOP, whose goal is finding an organization that minimizes the user's cognitive effort during data exploration. They proposed a metaheuristic named '*Organize*' with two neighborhood structures for the problem. However, they still need to provide an implementation for these.

Achieving more robust data discovery is possible by combining search and navigation (Nargesian et al., 2023). In this way, Ouellette et al. (2021) proposed RONIN as an exploration tool that integrates these two paradigms. In RONIN, a user can perform a keyword or joinability search over a data lake, then browse the result using an organization built by *Organize*. During navigation, the user can refine the query, retry the search, and return to navigating an updated organization based on the search. However, this requires generating organization in near real-time, which demands more efficient algorithms.

Previous works, except for Aso et al. (2020), infer a model from the data that enables its navigation. Some of them address the problem of searching datasets, as in Aso et al. (2020), Fernandez et al. (2018a,b), and Ouellette et al. (2021). However, only Nargesian et al. (2020, 2023) and Ouellette et al. (2021) propose approaches that infer data structures effective for navigation, and all of them use the same *Organize* algorithm to achieve this. Our work contributes to the more efficient induction of a more effective navigable structure for a data lake. Table 1.1 compares the features of our work and those.

Table 1.1: Comparison between main features of our work and similar literature proposals.

| References | Model Inference | Search Engine | Effective Browsing |
|---|---|---|---|
| Our proposed solution | ✓ | | ✓ |
| Aso et al. (2020) | | ✓ | |
| Fernandez et al. (2018b) | ✓ | ✓ | |
| Futia et al. (2020) | ✓ | | |
| Gupta et al. (2017) | ✓ | | |
| Nargesian et al. (2020) | ✓ | | ✓ |
| Nargesian et al. (2023) | ✓ | | ✓ |
| Ouellette et al. (2021) | ✓ | ✓ | ✓ |
| Torregrossa et al. (2021) | ✓ | | |
| Yu et al. (2021a) | ✓ | | |

Although Simulated Annealing is a traditional heuristic, it is still highly competitive in significant problems with a high cost of evaluating the quality of the solution (Ceschia and Schaerf, 2024). This performance is due to the possibility of exploring the solution space well without evaluating the entire neighborhood through local search. Danandeh Mehr et al. (2020) proposed a new hybrid model that enhances the Elman Neural Network (ENN) with an SA optimization for spatio-temporal drought forecasting. This work compares the performance of ENNs with and without SA, where the former predominates.

Osegi and Jumbo (2021) conducted a comparative analysis of Artificial Neural Network (ANN) algorithms for detecting credit card fraud and found that the SA-optimized model was superior, especially compared to a well-known robust model such as LSTM. To effectively deal with large instances in the capacitated facility location problem with customer incompatibilities, Ceschia and Schaerf (2024) introduced a Multi-neighborhood SA, which proved to be more ef-

fective than more exhaustive heuristics based on local search such as ILS, GRASP and the Evolutionary Algorithm.

# 2

# Technologies and Concepts

This Chapter reviews Big Data, presenting its opportunities and challenges. It also examines a leading solution for Big Data: the Data Lake. It offers the main features of this solution and discusses metadata management as a central challenge. Finally, it describes the data lake organization as a recent approach contributing to this issue.

## 2.1 Big Data

Data creates a modern way of interacting with the world, transforming how we live, work, and play. Companies and industries worldwide are using data to evolve their business models to become more efficient, improve the customer experience, and develop new sources of competitive advantage. People live in an increasingly digital world. They depend on online and mobile channels to connect with friends and family, access goods and services, and manage almost every aspect of their lives.

Data drives today's economy, and this dependency will only increase as services become more intelligent and data collection increasingly ubiquitous. Data generally comes from transactional systems, the Internet of Things (IoT), and social media (Reinsel et al., 2018). According to predictions, there are demands to connect 75 billion devices worldwide by 2025 (IRENA Group, 2019).

An integration scenario of heterogeneous devices is the expected setting for smart city environments (Ramos et al., 2023). Smart cities emphasize the interconnection between systems to produce data about themselves, allowing information sharing between different platforms. We can achieve the sharing with ubiquitous monitoring, data analysis, and homogeneous representation of all information in a unified framework (Rathore et al., 2016).

Furthermore, decision-makers can benefit from the information to plan how to use resources and expand smart areas (Bibri, 2018). Smart cities can leverage data applications to numerous

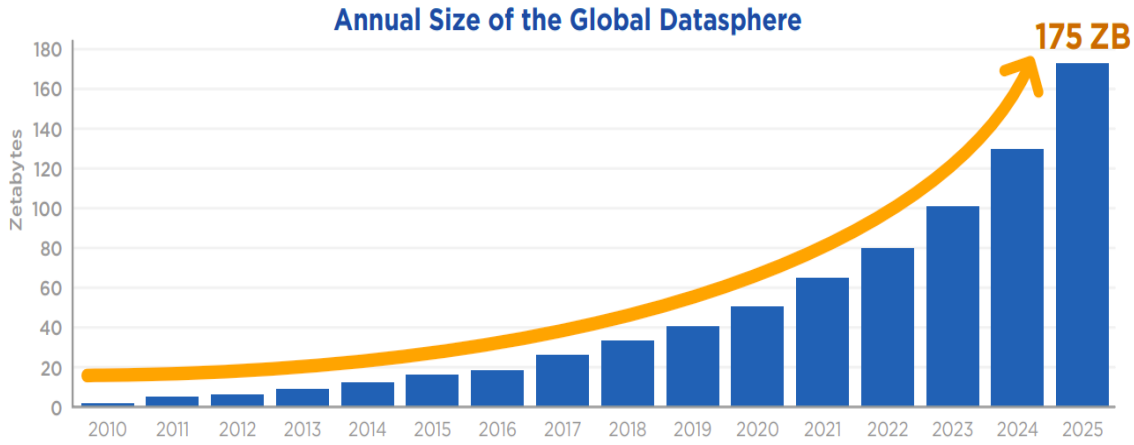



Figure 2.1: Annual Size of the Global Datasphere (Reinsel et al., 2018)

sectors, providing better customer experiences and services to increase performance for companies (e.g., increased profits and market value) and public institutions (e.g., reduced costs and increased customer satisfaction).

The consequence of this increasing reliance on data has been the exponential growth in the size of the Global Datasphere, as seen in Figure 2.1. The Global Datasphere quantifies the amount of data created, captured, and replicated in any given year across the world (Reinsel et al., 2018). Estimated to be 33 Zettabytes in 2018, IDC forecasts the Global Datasphere to grow to 175 Zettabytes by 2025 (Reinsel et al., 2018).

This scenario offers a complete set of opportunities. However, it requires storing and analyzing massive and complex data that exceeds the capacity of conventional computer systems and algorithms (Marx, 2013). This characteristic is the main feature of Big Data. Big Data is often defined by 5Vs (Stefanowski et al., 2017): (1) *Volume* (referring to the huge amount of data); (2) *Variety* (indicating multiple, heterogeneous, and complex data representations); (3) *Velocity* (referring to the speed to generate and analyze the data, as well as their dynamics and evolution in time, e.g., data streams); (4) *Veracity* (referring to the worse quality and uncertainty of data); (5) *Value* (referring to a potential business value that big data analysis could give).

Big Data is a reality nowadays. However, transitioning from previous approaches developed for knowledge discovery from databases to Big Data raises new challenges. Complexity not only affects the structure of data, but also the techniques for data collecting and analyzing. Although arguably advanced in recent decades, data processing and analysis methods are primarily designed for static and well-structured (usually tabular) data representations (Karimi et al., 2021).

In addition, there is an evident scarcity of approaches, algorithms, and technologies that explicitly address all the Vs, and the demand for them is growing steadily (Stefanowski et al., 2017). It is partially driven by years of practice-recurrence that have brought an avalanche of usage scenarios and applications that result in massive volumes of complex, poorly structured, and dynamic data in many domains, such as social networks, mobile services, network monitoring,

smart cities, and intelligent transportation.

The Data Warehouse is one solution that has dominated the Big Data landscape for years and is still very relevant. This solution has reached a high maturity level and is built on well-established technologies (mainly the Apache Hadoop environment toolset) (Alekseev et al., 2016). A Data Warehouse is a data repository from heterogeneous independent sources that have undergone a subject-oriented integration and structuring procedure (Chandra and Gupta, 2017). It provides tools to transform raw data, clean it, and integrate it into a unified storage repository for further analysis and decision support (Alekseev et al., 2016).

We model data warehouses multidimensionally to facilitate complex analysis and visualization (Chandra and Gupta, 2017). The processed and stored data follows a cube pattern in multidimensional databases (MDDBs), which organize the data into dimensions and fact tables (Chaudhuri and Dayal, 1997). Typically, the data originating from multiple data warehouse sources does not comply with its multidimensional model (Ali and Wrembel, 2017). For example, a long text document requires natural language processing to produce relevant topics and counts inserted into some fact or dimension table columns.

Furthermore, the incoming data may need to be more consistent and have better quality, presenting missing, redundant, and even contradictory values (Ali and Wrembel, 2017). Consequently, purpose-built software is commonly used in a data warehouse architecture to integrate its different sources (Machado et al., 2019). This software, or procedure, is known as Extraction–transformation–loading (ETL), and we use it as an intermediary between data sources and a data warehouse (Machado et al., 2019).

As limitations for this Big Data solution, we can mention resistance to change and adaptation, which usually impacts business models. There is a limitation of the available data due to the inability of the ingestion process to extract all available information and load it into a multidimensional model (Sawadogo and Darmont, 2021). This limitation utterly restricts the amount of information accessible to understand events and aid decision-making (Ramos et al., 2023). Thus, data warehouses become better options for applications with limited requirements in terms of variety and scalability.

## 2.2 Data Lake

Although researchers have discussed big data for some years, it still has many research challenges, especially the variety of data. It poses considerable difficulty to integrate, access, and query a large volume of diverse data in information silos with the traditional 'schema-on-write' approaches such as data warehouses (Stefanowski et al., 2017). Data lakes have been proposed as a solution to this problem. They are repositories storing raw data in their original formats and providing a standard access interface (Fernandes et al., 2023a).

Data Lakes offer flexibility for changing, storing, and integrating different structured, semi-

structured, or non-structured data sources. Considering the access gain and cost reduction of cloud services, the volume of data can increase more than enough to enable rich information results. It is also possible to perform batch processing and search through advanced Distributed Computing algorithms (He and Da Xu, 2012). Finally, table 2.1 presents some advantages of using a data lake or a data warehouse, with the characteristic of the systems playing a significant role in choosing the more appropriate or both solutions.

Table 2.1: Data Warehouse vs. Data Lake: advantages and disadvantages.

| | **Data Warehouse** | **Data Lake** |
|---|---|---|
| **Advantages** | Low maintenance<br>Matured technology<br>Optimized searches<br>Usability | Volume and variety<br>Flexible to changes<br>Batch processing<br>Advanced analysis<br>Governance |
| **Disadvantages** | Limited data<br>Resistance to change | High technical expertise<br>Complex maintenance |

We have seen the surge of many data lake architectures in recent years. Most of them use the zone concept. They assign the data to each zone according to its refinement degree (Giebler et al., 2019). For example, Figure 2.2 presents a general representation of a zone-based architecture composed of four zones to perform data Ingestion, Distillation, Processing, and Insights.

In addition, a governance layer traverses all the other layers to assure data security, privacy, quality, and monitoring (Giebler et al., 2019). Each layer has different governance levels, and the further from the ingestion layer, the greater the governance level (Giebler et al., 2019). However, this is only one of several variants of the zone architecture. Such architectures generally differ in the number and characteristics of the zones (Giebler et al., 2019).

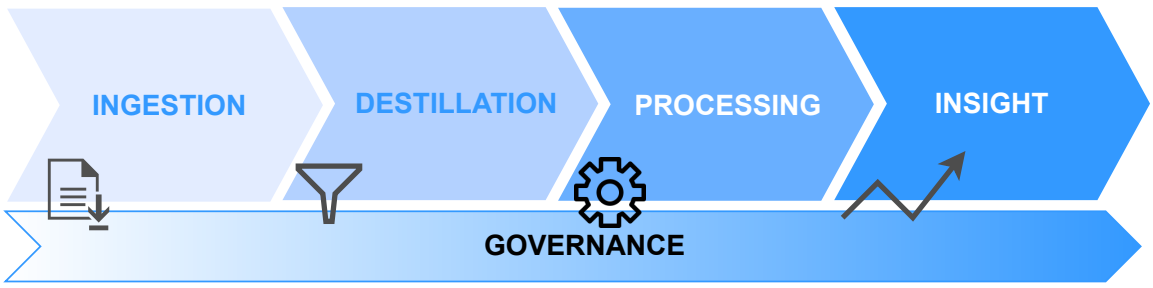



Figure 2.2: General data lake zone architecture.

## Ingestion

The data is stored in raw format or with minimum processing during ingestion. Also, integrating different applications creates a voluminous non-structured storage unity, highly connected and augmented with metadata (Sawadogo and Darmont, 2021). The aim is to minimize and avoid information loss by not prematurely processing data during the ingestion stage. The main features of data lakes lie in the late-binding aspect. Some differences between the mentioned approach and early-binding are presented in Table 2.2, adapted from Fang (2015).

Table 2.2: Key differences between early and late binding.

| | **Early-Binding** | **Late-Binding** |
|---|---|---|
| **Data Storage** | Well-defined schema; Ingest data<br>Processing; Evaluate data | Ingest data<br>Data catalog |
| **Users** | Consume data<br>Produce information | Define data schema; Processing<br>Consume data; Evaluate data<br>Produce information |
| **Suited for** | Known predictable sources<br>Reusability<br>Consistent results | Unfamiliar data<br>Infrequent usage<br>Heterogeneous sources |

Late-binding has no pre-defined schema. The input could be more familiar (structured or unstructured). The storage aims to ingest and save data processed when needed. Although there is no unique structuring, data must be well documented and associated with its metadata to facilitate discovery through the data lakes catalog. Users can serve themselves to process, evaluate, and extract information from diversified and heterogeneous sources (Gorelik, 2019). Early binding is well suited to known and predictable sources. Applications have access to treated, validated, and ready-to-use data consumed consistently to absorb the information provided by analysis, graphs, and results of inference models.

Usually, data lake architectures copy data from the source to a scalable or distributed file system such as HDFS (Hadoop Distributed File System) (Shvachko et al., 2010). There are known sets of technologies to transfer and process data, such as Apache Kafka (Le Noac'H et al., 2017), Sqoop (Vohra, 2016), Flink (Carbone et al., 2015), and Samza (Noghabi et al., 2017). We can mention a data table, an XML (Extensible Markup Language) file, or raw text as source examples.

The additional required storage is one of the disadvantages of these methodologies, as well as the need for periodic data updates (Stefanowski et al., 2017). We can avoid these problems with a federated data lakes approach, in which data is accessed directly in its source through distributed query engines (e.g., PrestoDB, Apache Drill, and Hive) connected to the databases or file systems (Mami et al., 2019). The performance loss is one of the issues of this approach since a search on multiple distinct servers can increase the complexity of queries. There are active research and cache mechanisms to mitigate the problem (Stefanowski et al., 2017).

Although there is barely any processing in the ingestion layer, and data is primarily found in its raw state, many problems and challenges arise from ample heterogeneous storage, especially from a data lake perspective. For example, how can a single query hit multiple databases and fetch relevant data from a similar context? For such cases, we can find previous research proposals on using semantic integration to match equivalent attributes (Li and Clifton, 1994; Li et al., 2000).

This process involves extracting semantics, expressing it as metadata, and matching equivalent data through discovery instead of pre-programming. For example, Facebook's recent efforts use natural databases to query data based on facts represented as short natural language sentences. One of the benefits of neural database systems is that they have no pre-defined schema. Another convenience is that users can perform updates and queries in natural language forms. For now, the success of neural databases depends on overcoming scalability challenges since we can not scale them to non-trivial databases nor apply them to set-based and aggregation queries (Thorne et al., 2021).

## Distillation

Distillation is the stage in which data is segregated and cataloged according to governance policies and other relevant criteria such as quality and security (Sawadogo and Darmont, 2021). Some segregation practices split data between free or restricted access. For example, smart cities allow access to health care, public security, and education applications. The distillation's granularity can be as fragmented as needed to facilitate search and satisfy governance requirements. After segregation procedures such as anonymization or access restrictions, we subject each data batch to treatment and validation.

Data scientists, engineers, analysts, and business managers access data at different refinement levels and have different perspectives. As the application's privacy policy demands, we should submit the data to segregation criteria before being available in the data lakes catalog. Data quality impacts application and business models, and treatment and validation steps aim to produce ready-to-go data to generate fast and accurate results (Gao et al., 2016). If specific data treatment and manipulation are required to boost results, users can access data in its raw state and apply their treatment and analysis.

## Processing

In the processing stage, users should be able to seamlessly manipulate data using automation tools such as Apache NiFi (Mathis, 2017). Big non-processed data give the user more freedom to apply techniques to select features, identifying hidden patterns that otherwise would be erased by premature processing. Future-proofing comes as another advantage since new projects will be able to access the data that initially had no purpose, minimizing the impacts of the fast-paced

evolution of big data (Gorelik, 2019).

With more possibilities to choose from, we can take advantage of feature engineering more broadly to maximize efficiency and information gained from algorithms (Heaton, 2016). Furthermore, since data lakes aggregate multiple data sources, the users can benefit from big-scale analysis in a structured platform that eases the overhead of experiment replication. We can embed previously recurrent tasks, such as data augmentation performed by data scientists, in the processing layer to speed up the deployment of machine learning and statistical inference models. Data augmentation provides a suite of techniques that enhance the size and quality of datasets (Shorten and Khoshgoftaar, 2019). We can store manipulated and enhanced data in the data lakes for future use and information harvesting.

We can use parallel data processing tools such as Apache Hadoop, which contains MapReduce (Sawadogo and Darmont, 2021) operations for large datasets. However, since MapReduce saves processing results in the computer's hard disk, it has performance limitations, making it inefficient for fast data (Li et al., 2018). To overcome such limitations, alternative frameworks such as Apache Spark, Apache Flink, and Apache Storm propose an in-memory approach rather than the file system to save intermediate results (Sawadogo and Darmont, 2021). We can implement Flink and Storm simultaneously as stream processing engines to real-time data (Li et al., 2018).

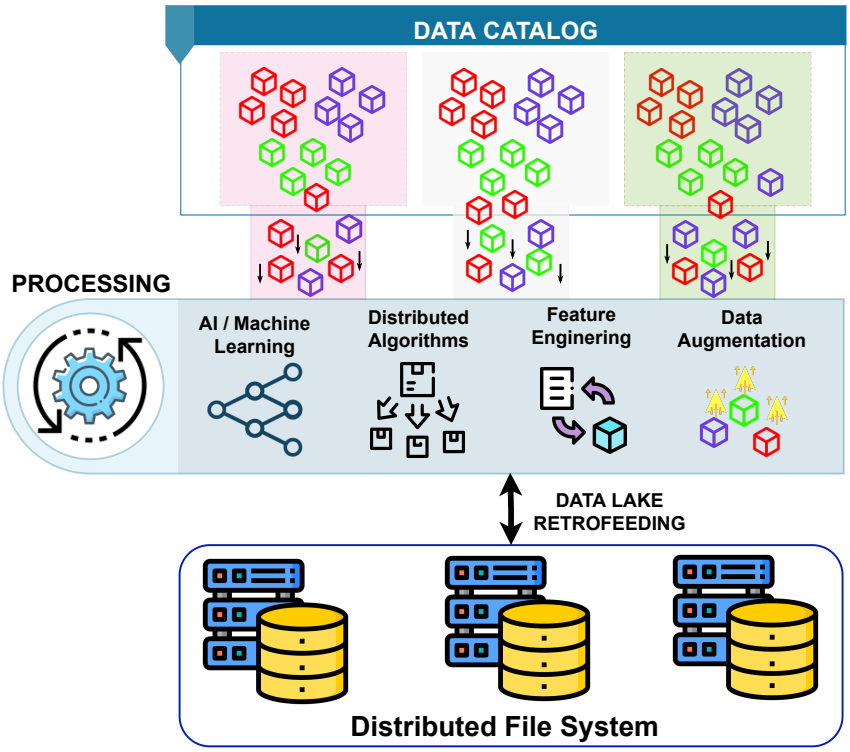



Figure 2.3: Data processing (Ramos et al., 2023).

## Insights

Most business ideas, problem identification, and solutions emerge from analyzing and reflecting on situations and results. So, the whole point of a data lake is to provide the feedstock data to fuel information gain. Therefore, users should have access to a framework to highlight the events of a specific business or application. This action occurs through an insights layer available to managers or stakeholders.

We use available analysis tools, such as Tableau, Microsoft Power BI, and Apache Zeppelin, to understand and expose event results to managers and stakeholders (Sawadogo and Darmont, 2021; Li et al., 2018). Such evaluations enable meaningful analysis and data-driven decision-making to continuously improve the success of a specific business or application (Provost and Fawcett, 2013).

The types of decisions that matter fall, more specifically, into two categories: i) decisions for which "discoveries" need to be made in the data; and, ii) repeatable decisions, especially on a large scale. Thus, decision-making can benefit even under small increases in the accuracy of this process based on data analysis. Data-driven decision-making enables decisions based on data analysis rather than intuition. The more data-driven it is, the more productive a company or government entity becomes (Provost and Fawcett, 2013). But it is worth mentioning that decision-makers need to be very cautious against decision bias (Kahneman et al., 2021).

## Governance

Several factors are constantly present to ensure good governance for every layer, preventing the emergence of vulnerabilities or turning the data lakes into a data swamp (Francia et al., 2021). Data swamps are unusable or hard-to-harvest datasets that only generate expenses to their maintainers, either by poor metadata management or by inaccessibility to users (Ramos et al., 2023). Data Lakes require meticulous care and monitoring, as they are complex storage solutions with advantages and disadvantages.

Data governance builds the foundation for data management and enforcement of the rules abided by stakeholders and decision-making processes. It covers the planning, monitoring, and execution of data lifecycle management and policies (Eichler et al., 2021). Lifecycle management refers to processes related to the design, creation, collection, storage, usage, maintenance, enhancement, archiving, and disposal of data (Eichler et al., 2021).

The policies cover all data lifecycle stages and involve security, privacy, compliance, and data quality management (Eichler et al., 2021). All involve generating or using stored information, or in this case, metadata (Francia et al., 2021). Considering a complex and heterogeneous system like a data lake, a robust metadata management system is imperative for optimized extraction of value and insights (Fernandes et al., 2023a).

As there is no main effort on data structuring, data lakes offer more flexibility to change

constantly, subject to governance policies of all data lake layers and access levels. The consistency assurance in multiple layers can make maintenance tasks more complex. The greater data volume and variety require more knowledge from users performing search and analysis. The decision to deploy a data lake depends on the number of integrated systems, heterogeneity, and integration degree.

## 2.3 Data Lake Organization

The rising popularity and size of data lakes are fueling interest in dataset discovery. The dataset discovery process is usually done by search querying or browsing. In search, the query can consist of a set of keywords or data, and the goal is to find datasets related to it (Brickley et al., 2019; Zhu et al., 2016). When the query is a tabular dataset, it is still possible to search for joinable and unionable tables (Bharadwaj et al., 2021; Nargesian et al., 2018).

In the browsing paradigm, a user navigates through an organizational structure to find data of interest. Such a mechanism allows expert Data Lake users to browse available topics, going from general concepts to more specific ones (Nargesian et al., 2020). This approach saves time while exploring the available data and in vaguely defined searches since it is unnecessary to check everything stored.

Existing taxonomies or structures provide hierarchical navigation over entities automatically created using taxonomy induction (Torregrossa et al., 2021; Gupta et al., 2017). Entities may have some known features to apply faceted-search over the entities (Aso et al., 2020). Taxonomy induction identifies *is-a* relationships between entities (e.g., Tesla *is-a* Company), whereas face search evaluates predicates (e.g., model = Cybertruck) to filter collections of entities (Kharlamov et al., 2017).

In contrast to hierarchies over entities, in data lakes, the domain of each data source can be related to many different types of entities. There may be no *is-a* relationships between datasets or their attributes and no easily defined facets for grouping them. A common approach is to annotate tables with class labels from a knowledge base (Mami et al., 2019). In this way, the *is-a* relationships between the class labels provide a kind of organization for the tables (Futia et al., 2020). However, knowledge bases are not designed to provide effective navigation (Nargesian et al., 2023).

Nargesian et al. (2020) proposes to build an organization designed best to support navigation and exploration over tables in data lakes. They define an organization as a Directed Acyclic Graph (DAG) with nodes representing sets of attributes from a data lake. A node may have a label created from attribute values or metadata. A table is associated with all nodes containing any of its attributes. An edge indicates that the attributes in a parent node are a superset of the attributes in a child one. Thus, a user can find a table traveling from an organization source node to any sink node containing its attributes.

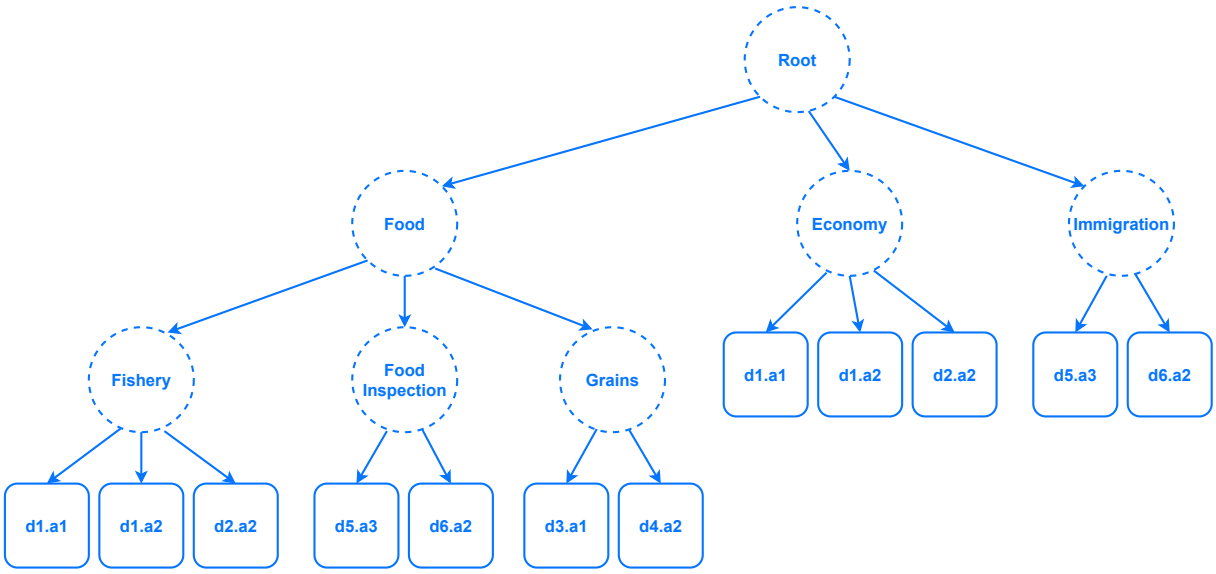



Figure 2.4: An organization for the data lake described in the Table 2.3, as provide by Nargesian et al. (2023)

For example, consider the collection of tables from an open data lake described in the table 2.3. A table can be multi-faceted, and its attributes can regard different topics. One path to reveal this data to a user is through a flat structure of all attributes. A user can browse the data and select data of interest. With many tables, an organization over attributes may provide more efficient navigation. Suppose the data lake organization is in the DAG of Figure 2.4. The label of a non-sink node in this organization summarizes the content of the attributes in its subgraph. Suppose a user is interested in the topic of food inspection. Using this organization, the number of options at each navigation step is limited to three, and only two steps are necessary to achieve the goal.

Table 2.3: Sample Table from Open Data (Nargesian et al., 2023)

| Id | Table Name |
|---|---|
| d1 | *Surveys Data for Olympia Oysters, Ostrea lurida, in BC.* |
| d2 | *Sustainability Survey for Fisheries.* |
| d3 | *Grain deliveries at prairie points 2015-16.* |
| d4 | *Circa 1995 Landcover of the Prairies.* |
| d5 | *Mandatory Food Inspection List.* |
| d6 | *Canadian Food Inspection Agency (CFIA) Fish List.* |
| d7 | *Wholesale trade, sales by trade group.* |
| d8 | *Historical releases of merchandise imports and exports.* |
| d9 | *Immigrant income by period of immigration, Canada.* |
| d10 | *Historical statistics, population and immigrant arrivals.* |

Nargesian et al. (2020) define the Data Lake Organization Problem as finding an organization that allows a user to most efficiently find tables. They describe the user navigation through an organization using a Markov model. In this model, each node in an organization is equivalent to

a state in the navigation model. They provide a set of the following states (the children of the current) to a user in each state. An edge denotes the transition from the current state to the next state. Due to the subset property of edges, each transition filters out some attributes until the browsing reaches characteristics of interest.

The effectiveness of an organization is defined as dependent on a set of properties. The first of these is related to the branching factor. The number of choices presented to the user at each step should be reasonable. Another consideration is the distinction between the options available in each state. This aspect makes it easier for users to choose the most relevant one. Additionally, the transition probability function of the proposed Markov model assumes that users choose the next state that has the highest similarity to the topic query they have in mind. Finally, it is established that the number of choices they need to make (the length of the discovery path) should be insignificant.

# 3

# Methodology

This Chapter reviews the definition of the Data Lake Organization Problem. It also discusses the best algorithm proposed in the literature for this problem. Finally, we offer multi-start simulated annealing for the DLOP and algorithms for neighborhood structures defined by Nargesian et al. (2020).

## 3.1 Data Lake Organization Problem

To provide a complete understanding of the Data Lake Organization Problem, this section summarizes the concepts, formulations, figures, and definitions presented by Nargesian et al. (2020, 2023) in their seminal works.

### 3.1.1 Organization

Let $\mathcal{T}$ be the set of all tables in a data lake. Each table $T \in \mathcal{T}$ consists of a set of data columns named attributes and denoted by $attr(T)$. Let $\mathcal{A} = attr(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} attr(T)$ be the set of all attributes in a data lake. For example, with the data lake in TABLE 2.3, we have $\mathcal{A} = \{d1.a1, d1.a2, d2.a1, d3.a1,\ d4.a1, d5.a1, d6.a1, d7.a1, d8.a1, d9.a1, d9.a2, d10.a1\}$. As defined in Nargesian et al. (2020), an organization of $\mathcal{T}$ is an agglomerative clustering of $\mathcal{A}$ in a directed acyclic graph (DAG). Let $O = (V, E)$ be an organization of $\mathcal{T}$, then every $v \in V$ corresponds to a set of columns $D_v \subseteq \mathcal{A}$. For instance, in the organization in Fig. 1, the node `Fishery` contains the attributes $D = \{d1.a1, d1.a2, d2.a1\}$. Every column $A \in \mathcal{A}$ has a set of values called the domain ($dom(A)$). The domain of a node $v$ is $dom(v)$, which is $dom(A_v)$ when $v$ is a sink node and $\bigcup_{A \in D_v} dom(A)$ otherwise.

A vertex $u$ is a parent of $v$ if $uv \in E$. The vertex $v$ is then a child of $u$. A vertex $u$ is an ancestor of $v$ if the digraph contains a path from $u$ to $v$. If $u$ is an ancestor of $v$, then $v$ is a descendant of $u$. For example, `Fishery` is the child of `Food` and descendant of `Root` in Fig. 2.4.

Let $ch(\cdot)$ map a node to its children, and $par(\cdot)$ be the mapping of a node to its parents. Every edge $uv$ indicates a relationship such that $D_v \subseteq D_u$. This is called *inclusion property*, and implies $\forall v \in V, D_v = \bigcup_{w \in ch(v)} D_w$. For instance, the `Food` node contains the union of all columns under `Fishery`, `Food Inspection`, and `Grains`. Hence, each sink node $z$ corresponds to a distinct attribute $A_z \in \mathcal{A}$. An organization has only one source node, called `Root`, and it originates all user navigation. The `Root` corresponds to the set of all columns $\mathcal{A}$.

### 3.1.2 Navigation Model

The user experience during an organization's discovery, Nargesian et al. (2023) consider a model based on the Markov chain. Each node corresponds to a state indicating the user's position, and each edge corresponds to a transition. For simplicity, we will use these related terms interchangeably. The navigation starts in the source node, and the user selects the next node at each browsing step. Consequently, because of the inclusion property, the transition state filters out some attributes of the previous node. The discovery process stops once the user reaches a sink node.

The user's intent is modeled as a search topic $X$. Given an organization $O$ and a search topic $X$, the transition probability $P(c|s,X,O)$ regards the likelihood that the user who searches for $X$ will choose $c$ as the next state if they are at the state $s$. The probability should rely on the similarity between $dom(c)$ and $X$, so let $\kappa(c,X)$ be a similarity metric. The transition probability is

$$P(c|s,X,O) = \frac{e^{\frac{\gamma}{|ch(s)|} \cdot \kappa(c,X)}}{\sum_{t \in ch(s)} e^{\frac{\gamma}{|ch(s)|} \cdot \kappa(t,X)}}. \tag{3.1}$$

The term $|ch(s)|$ is a penalty factor to avoid excessive choices in each browsing step. The impact of the high similarity of a state with $X$ reduces when its parent has a high branching factor. To adjust the importance between the similarity to $X$ and the out-degree, a similarity factor $\gamma$ is included. Such a constant must be a strictly positive number. Its relevance is evident when the $\kappa(\cdot,\cdot)$ image has a too-short range relative to the desired branching factor.

The goal of exploring the content of heterogeneous data lakes makes it reasonable to employ a semantic similarity. Consequently, the literature has chosen to build organizations on text attributes. To capture semantics, each data value $v$ is represented by an embedding vector $\vec{v}$ such that values more likely to share the same context have embedding vectors close in embedding space according to some metric. Every state $s$ is indicated by a topic vector, which is the sample mean ($\mu_s$) of set $\{\vec{v}|v \in dom(s)\}$; the transition probability to state $c$, $\kappa(c,X)$ is the cosine similarity between $\mu_c$ and $\mu_X$.

All states in $O$ are reachable from the `Root` through some discovery sequence. A discovery sequence is a path, $r = s_1, \ldots, s_k$ where $s_i \in ch(s_{i-1})$ for $1 < i \leq k$. The Markov property claims

that the probability of transitioning to a state depends only on its parent. Hence, the probability of reaching state $s_k$ through a discovery sequence $r = s_1, \ldots, s_k$, while looking for $X$ can be expressed as

$$P(s_k|r,X,O) = \prod_{i=2}^{k} P(s_i|s_{i-1},X,O). \tag{3.2}$$

This formulation naturally punishes longer sequences. Since an organization is a digraph, a state is reachable by multiple discovery sequences. Let $paths(s)$ be the set of all discovery sequences in $O$ that achieve $s$ from the `Root`. The probability of reaching a state $s$ in $O$ while searching for $X$ is

$$P(s|X,O) = \sum_{r \in paths(s)} P(s|r,X,O), \tag{3.3}$$

which is equivalent to

$$P(s|X,O) = \sum_{p \in par(s)} P(s|p,X,O)P(p|X,O). \tag{3.4}$$

### 3.1.3 Organization Discovery Problem

Finding an appropriate organization for a data lake is achieved by optimizing a DAG $O$ considering the topics held by its tabular attributes. Therefore, the organization construction always assumes $X \in \mathcal{A}$. The discovery probability of an attribute $A$ in organization $O$ is $P(s_A|A,O)$, where $s_A$ is a sink node. We denote the discovery probability of $A$ as $P(A|O)$.

The discovery of a table $T$ in an organization is $O$ whenever a discovery sequence reaches any of its attributes. As there is independence for finding features, the discovery probability of a single table $T$ is

$$P(T|O) = 1 - \prod_{A \in T} (1 - P(A|O)). \tag{3.5}$$

For a set of tables $\mathcal{T}$, the organizational effectiveness is the expected probability of finding tables, defined as

$$P(\mathcal{T}|O) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} P(T|O). \tag{3.6}$$

Finally, given a set of tables $\mathcal{T}$ in a data lake, the **Data Lake Organization Problem** is to find an organization $\hat{O}$ such that

$$\hat{O} = \arg\max_{O} P(\mathcal{T}|O). \tag{3.7}$$

## 3.2 Data Lake Organization Through Multi-Start Simulated Annealing

Simulated Annealing is one of the best-known metaheuristic methods for addressing challenging global optimization problems (Delahaye et al., 2018). SA is a good candidate when there is no objective function explicitly, and its evaluation involves complex simulation processes requiring a lot of memory (Delahaye et al., 2018). SA proves to be more powerful than traditionally more robust metaheuristics (i.e., GRASP, ILS, and GA) in problems where generating neighboring solutions and calculating fitness is extremely costly in terms of time and space (Ceschia et al., 2022; Ceschia and Schaerf, 2024). We achieve better results when the SA is enhanced with a multi-start strategy (Yu and Lin, 2014), which is motivated by its capacity to improve exploration and overcome local optima. This approach mitigates sensitivity to initial solutions, handles stochasticity, and adapts to diverse problem instances (Heu, 2010).

Our Data Lake Organization proposal uses a multi-start strategy and considers as input the data lake structure $\mathcal{T}$ and different hyper-parameter: similarity factor $\gamma$, temperature decay factor $\alpha$, and several explored solutions $k_i$ until the temperature decreases. To find the best organization, we consider the Algorithm 3.1 based on a multi-start strategy. The algorithm considers the following steps: (1) Creation of an initial organization (section 3.2.1); (2) Application of the Simulated Annealing process $k_r$ times to improve initial solution during $k_f$ interactions (section 3.2.2); (3) Choice the organization with the highest effectiveness according to Eq. 3.6.

Due to the complexity of the problem, our approach has been modulated into a set of algorithms whose interdependence and overall execution flow are illustrated in Figure 3.1. The arrows indicate that the source module (algorithm) executes the destination one. The double arrows show that this execution happens often until it reaches a stop condition. The diamond indicates a choice, a bifurcation between two possible execution flows. In general terms, our proposal starts by running the already discussed Algorithm 3.1, which runs Simulated Annealing (Algorithm 3.2) several times until that reaches the stop condition. Each run of this will evaluate multiple organizations generated by Random Neighbor Selection (Algorithm 3.3). This strategy performs a heuristic method to select a node from the graph and randomly picks an operation to modify it, which can add or delete a parent.

The Add Parent Operation (Algorithm 3.4) heuristically inserts a new parent node into the chosen node and then calls Update ancestors (Algorithm 3.5). This strategy will traverse the nodes that reach the changed node (in other words, its ancestors) and, for each of them, it will execute the Update state (Algorithm 3.6) until there are no more ancestors to update their domain information. Once these changes have been made, Algorithm 3.5 runs Update descendants (Algorithm 3.7), which downward traverses the graph updating the probabilities. At the end of this procedure, we have the sink nodes with attribute discovery probabilities available to compute the organization's effectiveness. The delete parent operation (Algorithm 3.8) heuristically removes a

parent (and also its siblings) from the chosen node, previously transferring their children nodes to their parents, and then calls Update descendants. Once the probabilities have been refreshed, we have a new organization ready to be assessed by the SA.

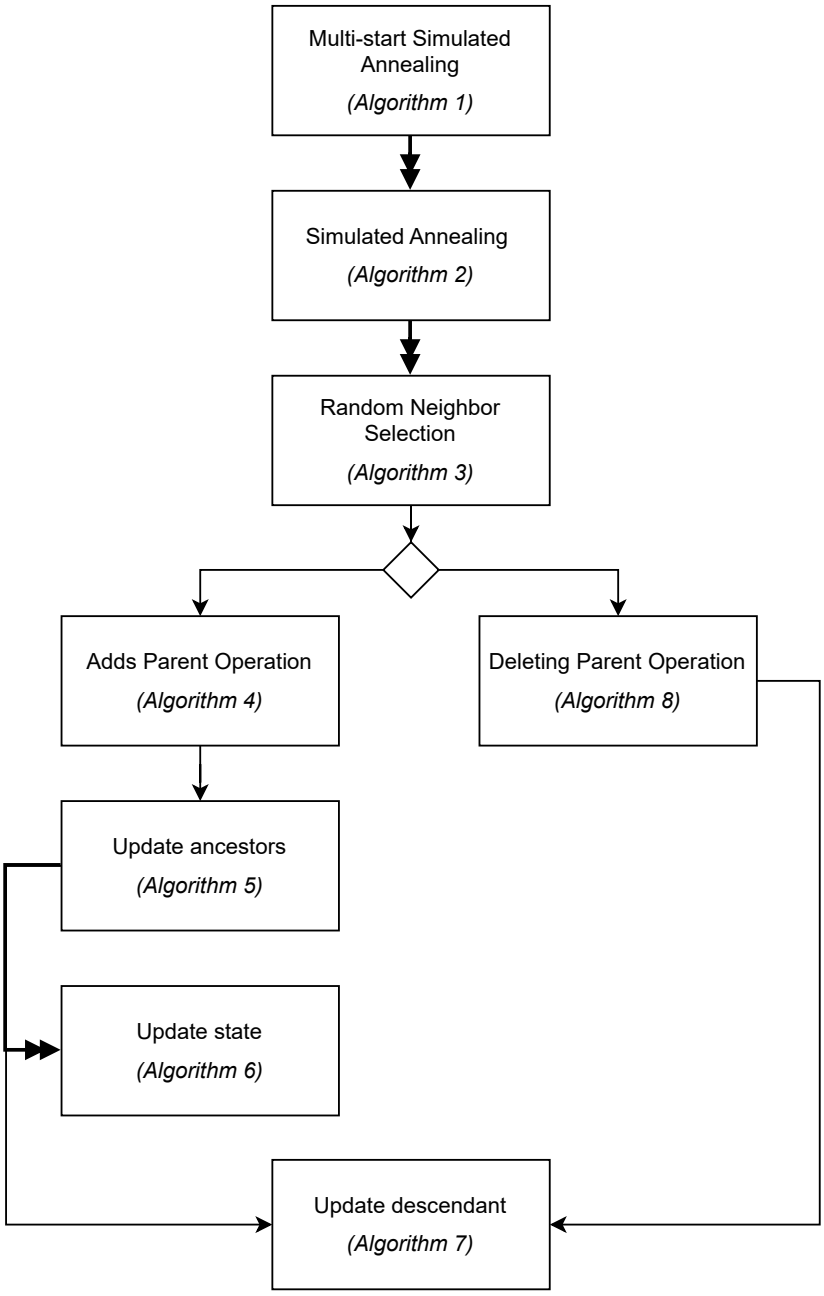



Figure 3.1: Flowchart of the proposed approach in terms of its component algorithms

### 3.2.1 Organization Initialization

The initial organization $\mathcal{O}$ is an induced tree from the attributes of the data lake, based on hierarchical clustering algorithms (Nargesian et al., 2020, 2023). In our case, the initialization relies on the nearest neighbor chain (NNC) algorithm designed for agglomerative hierarchical clustering,

**Algorithm 3.1** Multi-start Simulated Annealing

**Require:** $\mathcal{T}$, $\gamma$, $\alpha$, $k_r$, $k_i$, $k_f$
**Ensure:** $O^*$
$O \leftarrow$ `InitializeOrganization`$(\mathcal{T})$
**while** stopping criteria **do**
    $O' \leftarrow$ `SimulatedAnnealing`$(O, \mathcal{T}, \gamma, \alpha, k_i, k_f)$ ▷ Algorithm 3.2 in subsection 3.2.2
    **if** $O^* = \emptyset$ **or** $P(\mathcal{T}|O') > P(\mathcal{T}|O^*)$ **then**
        $O^* \leftarrow O'$
    **end if**
**end while**

whose complexity is $O(n^2)$ (Yu et al., 2021b). Over these clusters, in this work, we consider the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) to evaluate the distance between clusters (Murtagh and Contreras, 2011). In this strategy, the elements are the attributes' topic vectors, and the distance is the normalized inverse of the cosine similarity. At each iteration, the NNC creates a new node by agglomerating two others, decreasing the number of nodes available to cluster by one. Thus, there will be $|\mathcal{A}| - 1$ iterations, and the clustering will produce a graph with $V = 2 \cdot |\mathcal{A}| - 1$ nodes.

We create edges between the new state and the states of clustered nodes, and the former henceforth contains the union of their attributes. To save computation time and memory space, we store only the topic vector for each state instead of all the vectors that compose its domain. The topic vector is the sum of its embedding domain for each organization node. Hence, the topic vector in a non-sink node is the sum of the topic vectors in its children when their domains are disjoint. This strategy adapts the initial approach and does not distort the transition probabilities. Suppose a state $s \in V$ with a domain $dom(s)$. Let $dom'(s) \subset \mathbb{R}^d$ be the transformation image of $dom(s)$ for a given embedding space, and $\mu_s \in \mathbb{R}^d$, its sample mean. Then, for any query topic $X$ represented by $\mu_X \in \mathbb{R}^d$, we have

$$\begin{aligned} \kappa(s,X) &= \cos(\mu_s, \mu_X) \\ &= \frac{< \mu_s, \mu_X >}{\|\mu_s\| \|\mu_X\|}. \end{aligned} \tag{3.8}$$

Then,

$$\begin{aligned} \kappa(s,X) &= \frac{\left\langle \frac{1}{|dom(s)|}\sigma_s, \mu_X \right\rangle}{\frac{1}{|dom(s)|}\|\sigma_s\| \|\mu_X\|} \\ &= \frac{< \sigma_s, \mu_X >}{\|\sigma_s\| \|\mu_X\|}. \end{aligned} \tag{3.9}$$

Furthermore,

$$\kappa(s,X) = \cos(\sigma_s, \mu_X), \tag{3.10}$$

where $\sigma_s$ is the summation of the vectors in $dom'(s)$.

### 3.2.2 Simulated Annealing

After the organization initialization, we have the tree $O$ as the initial organization. Our Simulated Annealing approach considers as input the organization $O$ to be optimized, and the same hyper-parameter used in Algorithm 3.1. For the optimization of this initial organization structure, we consider the Algorithm 3.2 with the following steps: (1) Generation of new random solution based on the neighborhood of the current one (Delahaye et al., 2018); (2) If the new solution is better than the current one, it is accepted, and the search process resumes from this new current solution; (3) We adopt a worse solution than the current solution with some probability;

**Algorithm 3.2** Simulated Annealing with Early Stop

```
1: procedure SIMULATEDANNEALING(O, T, γ, α, k_i, k_f)
2:     A ← attr(T)
3:     O* ← O
4:     t ← P(O|T)
5:     k ← 0
6:     while t > 0 or k < k_f do
7:         for i ← 1 to k_i do
8:             O' ← RandomNeighbor(O, γ, A)        ▷ Algorithm 3.3 in subsection 3.2.2
9:             k ← k + 1
10:            Δ ← P(O|T) − P(O'|T)
11:            if Δ < 0 then
12:                O ← O'
13:                if P(O'|T) > P(O*|T) then
14:                    O* ← O'
15:                    k ← 0
16:                end if
17:            else
18:                x ← Random([0, 1))
19:                if x < e^{−Δ/t} then
20:                    O ← O'
21:                end if
22:            end if
23:        end for
24:        t ← αt
25:    end while
26:    return O*
27: end procedure
```

The differential in our implementation is the random neighbor selection. Instead of choosing a neighboring solution in a purely random manner, we use a roulette wheel selection approach (Goldberg, 1989). We consider neighborhood structures the addition and deletion of parents detailed in Section 3.2.3. To produce a neighbor organization, we consider the Algorithm 3.3 with the following steps: (1) The neighborhood structure is randomly selected. The procedures `AddParent` and `DeleteParent` (Algorithms 3.4 and 3.8) compose the neighborhood structure; (2) A roulette wheel selects an organization level with a probability inversely

proportional to its value; (3) It is choosing a state belonging to the drawn level with a probability inversely proportional to its reachability, which is defined as (Nargesian et al., 2023): $P(s|O) = \frac{1}{|\mathcal{A}|}\sum_{A\in\mathcal{A}} P(s|A,O)$. (4) The drawn operation (`AddParent` or `DeleteParent`) is applied to the chosen state, producing an organization neighboring the current one. This approach favors states closer to the `Root` and has a lower reachability probability. Then, their modification has more significant potential to impact the organization's effectiveness positively (Nargesian et al., 2023).

**Algorithm 3.3** Random Roulette Wheel Neighbor Selection

**procedure** RANDOMNEIGHBOR($O$, $\gamma$, $\mathcal{A}$)
  $h \leftarrow \sum_{l=2}^{depth(O)} l^{-1}$
  $l \leftarrow 2$
  $z \leftarrow 0$
  $x \leftarrow Random([0,h))$
  **while** $z < x$ **do**
    $z \leftarrow z + l^{-1}$
    $l \leftarrow l + 1$
  **end while**
  $\mathcal{S} \leftarrow states(O,l)$
  $h \leftarrow \sum_{\forall s\in\mathcal{S}} P(s|O)^{-1}$
  $z \leftarrow 0$
  $s \leftarrow$ **null**
  $x \leftarrow Random([0,h))$
  **for all** $s' \in \mathcal{S}$ **do**
    $z \leftarrow z + P(s'|O)^{-1}$
    **if** $z \geq x$ **then**
      $s \leftarrow s'$
      **break**
    **end if**
  **end for**
  $x \leftarrow Random([0,1))$
  **if** $x < 0.5$ **then**
    $O' \leftarrow$ `AddParent`$(O,l,s,\gamma,\mathcal{A})$ ▷ Algorithm 3.4 in subsection 3.2.3
  **else**
    $O' \leftarrow$ `DeleteParent`$(O,l,s,\gamma,\mathcal{A})$ ▷ Algorithm 3.8 in subsection 3.2.3
  **end if**
  **return** $O'$
**end procedure**

### 3.2.3 Neighborhood Structures

Similar to Nargesian et al. (2023), we restrict our choices of a new organization at each step to those created by the operations add and delete the parent. For add parent operation, given a selected state $s$ at level $l$, we insert a transition between $s$ and the state at level $l-1$ with the highest reachability probability that is not a parent of $s$. The new parent of $s$ and its ancestors are updated to contain the attributes in $s$, $D_s$, satisfying the inclusion property. Such an operation potentially increases the reachability probability of a state by adding more discovery paths ending at that state at the cost of increasing the branching factor (Nargesian et al., 2023).

Consider the organization presented before in Fig. 2.4. Suppose the selection of `Grains` node (level 2) for the operation, and `Economy` is the node with the highest reachability probability of level 1. Then, we will insert an edge from `Economy` to `Grains`, as shown in Fig. 3.2. Since the `Economy` did not contain the attributes held by `Grains`, the former must be updated to include them. Since the parent of `Economy` is the source node containing all the attributes, changing it is unnecessary. However, changing the attributes of the `Economy` implies a change in its domain and, consequently, in its topic vector.

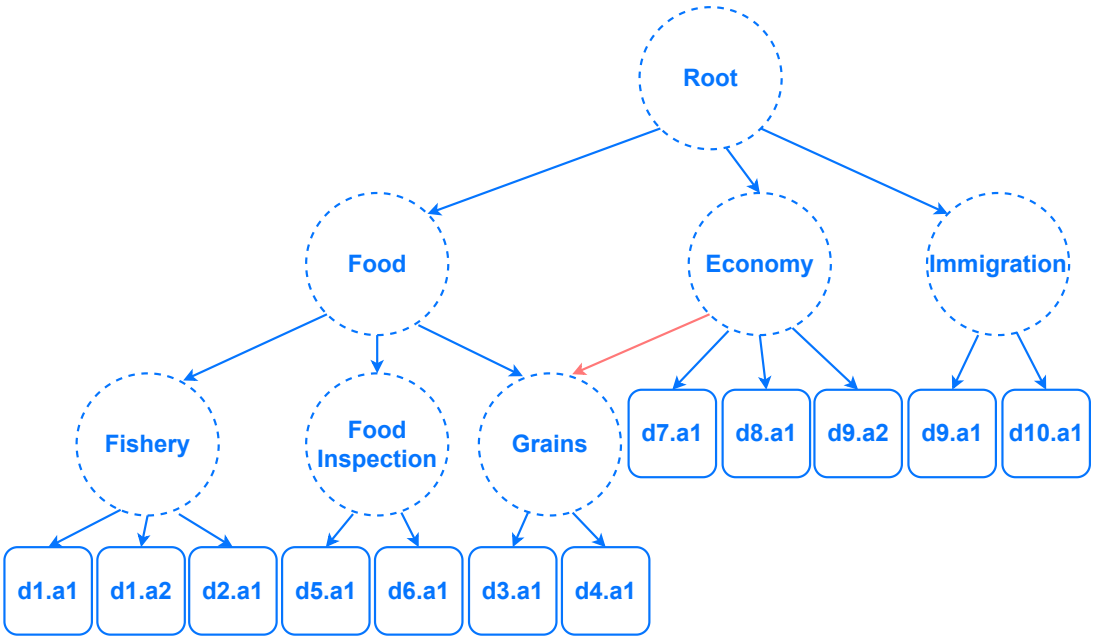



Figure 3.2: Add parent operation between `Economy` and `Grains` states.

Since the topic vector of a state impacts the transition probabilities of this state and its siblings (Eq. (3.1)), it is necessary to update the transitions that start from the source node. However, this implies the need to update the probabilities of all states reachable from the `Root` as a consequence of Eq. (3.4). For the same reason, updating all the probabilities of the edges that start from the `Economy` is also required. Fig. 3.3 illustrates this behavior, where the states and transitions updated due to adding parent operation are in red. Also, `Economy` becomes `Economy Grains` and is on a solid disk to indicate the change of its attributes and topic vector.

The delete parent operation removes the parent with the most negligible reachability probability of a given state $s$. We also remove the siblings of such a parent node to decrease the organization's length. With this, all nodes that have had their parents deleted become children of the states that were previously their grandparents. This operation does not change the graph path number that leads to the node, and due to the inclusion property, it does not change the representation of the grandparent nodes either. It only makes the length of paths to $s$ smaller, which boosts the reachability probability of $s$ (Nargesian et al., 2023). However, this increases the branching factor of such ancestors, which decreases their transition probabilities overall (Nargesian et al., 2020).

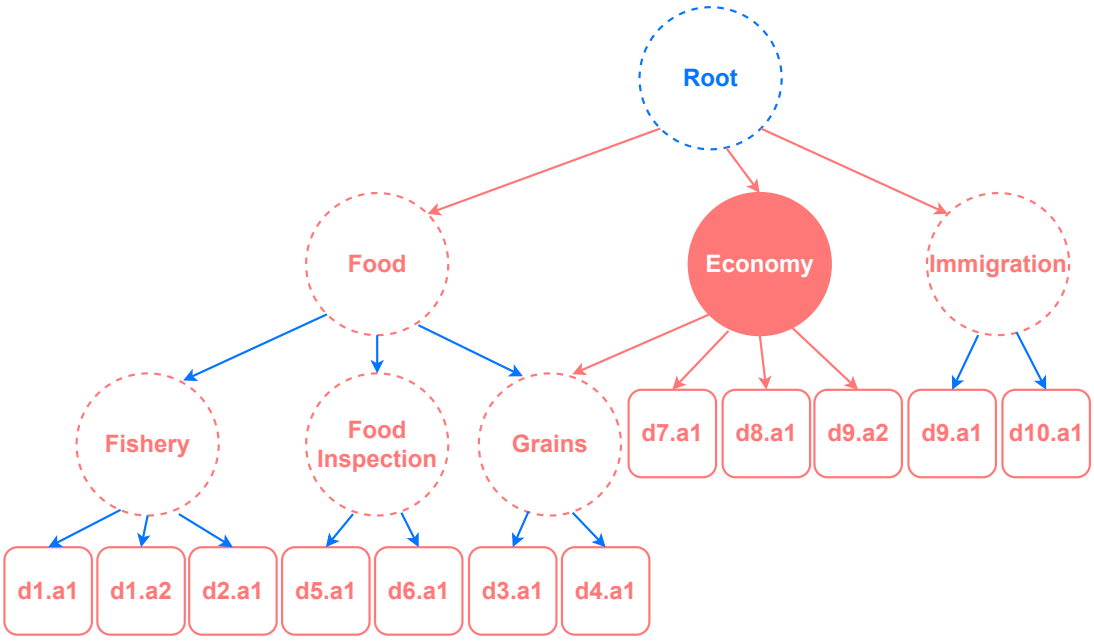



Figure 3.3: States and transitions that need updating after adding parent operation.

Considering Fig. 2.4, suppose that we delete the non-sink states belonging to level 2 (Fig. 3.4). By removing them and transferring their children to their parents, the `Food` state becomes the parent of all sink nodes that were their children. Since its attributes remain the same, this strategy does not change the `Food` domain. However, it shortens the path to the sink nodes reachable by `Food`, increasing the number of children. Therefore, updating the probabilities of `Food`'s descent is necessary, as illustrated in Fig. 3.5.

The `AddParent` procedure performs the parent addition, as presented in Algorithm 3.4, by the following steps: (1) Get the selected state $s$ and the set of states from the previous level; (2) Ruled out the states already parents or reachable from $s$. Such verification is necessary to avoid cycle insertion into the DAG; (3) The new parent is the remaining state with the highest reachability probability $p^*$, and from this is inserted an outcoming edge to $s$. (4) Finally, `UpdateAncestors` is executed to identify the nodes impacted by the operation and update them, keeping the organization's properties.

During the adds operation, we must update the attribute set of $p^*$ such that $D_s \subset D_{p^*}$ to preserve the inclusion property. In addition, we must recursively verify this into its ancestors. Since the domain determines the topic vector, this operation can potentially modify the transition probabilities of states very close to the `Root` (Nargesian et al., 2023). These modifications would update the probabilities in almost every state in the organization. Our update operation starts from $p^*$ and upward traverses the graph in breadth as it updates its ancestors' reach, attributes set, and topic vector.

Considering the inclusion property, we need only to compare the evaluated states with $s$, which does not require any specific sorting in the navigation. Nevertheless, we employed two pruning strategies in the graph traversal. The first considers that when visiting a node that does

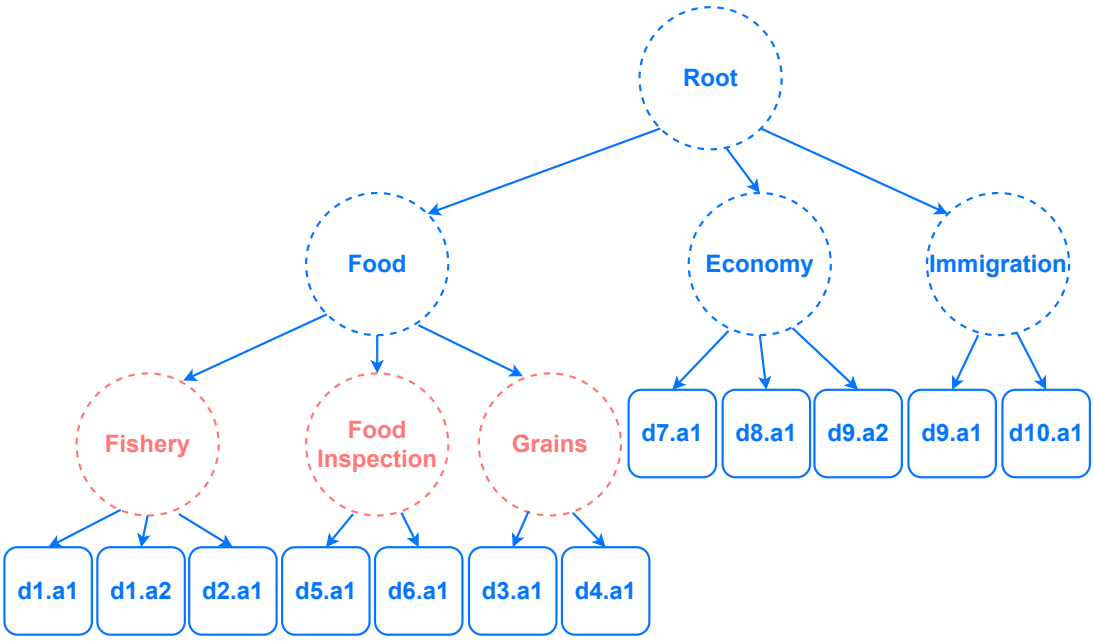



Figure 3.4: Delete parents applied to level 2 states.

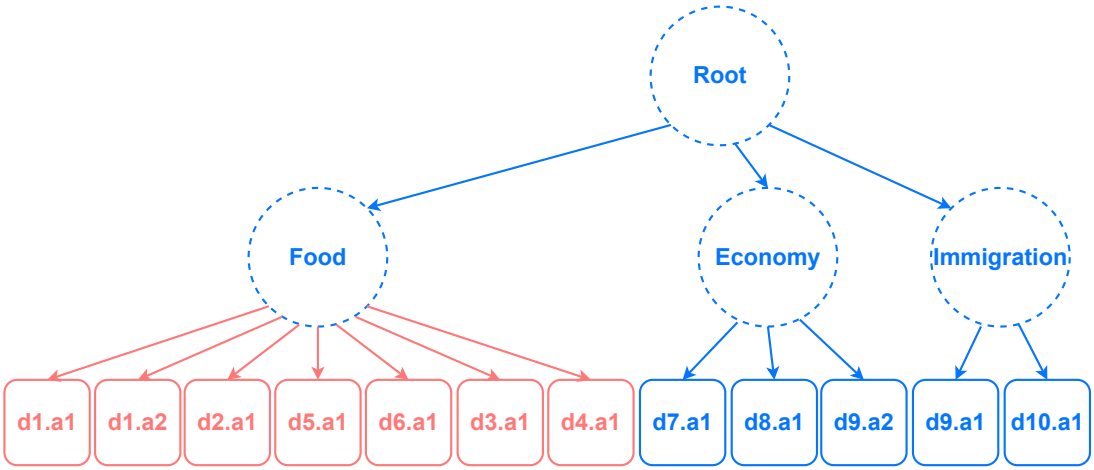



Figure 3.5: States and transitions that need updating after deleting parent operation.

not need a domain or reach update, its parents do not need to be evaluated by the inclusion property. The second pruning is related to the downward traversal needed to update the probabilities. In this strategy, the probabilities update starts from nodes that did not modify in their domains, unlike at least one of their children. It prevents this second update step from starting closer to the source node due to the reach update.

Such an update operation is performed by `UpdateAncestors`, as presented in the Algorithm 3.5, following the steps: (1) We insert the new parent $p^*$ in pair with a boolean true value into a queue $Q$. This value indicates whether we must update the state if we have the probabilities; (2) Then, we will iteratively remove the first node in the queue and the reach, attribute set, and topic vector updating by `UpdateState` operation. Such an operation returns the performed modification: none, reach, or domain (the last also includes reaching change); (3) When there is a

**Algorithm 3.4** Adds Parent Operation

**procedure** ADDPARENT($O$, $l$, $s$, $\gamma$, $\mathcal{A}$)
  $\mathcal{P} \leftarrow states(O, l-1)$
  $p^* \leftarrow$ **null**
  **for all** $p \in \mathcal{P}$ **do**
    **if** $ch(p) \neq \emptyset$ **and** $p \notin par(c)$ **and not** $reach(c,p)$ **and** ($p^* =$ **null or** $P(p^*|O) < P(p|O)$) **then**
      $p^* \leftarrow p$
    **end if**
  **end for**
  **if** $p^* \neq$ **null then**
    $E \leftarrow E \cup \{p^* c\}$
    `UpdateAncestors`$(c, p^*, \gamma, O, \mathcal{A})$ ▷ Algorithm 3.5 in subsection 3.2.3
  **end if**
  **return** $O$
**end procedure**

change (other than none), we insert the current node's parents into $\mathcal{Q}$ paired with a boolean flag; (4) If the domain changed, the flag will have the same value as the actual node; (5) Otherwise, the flag will be set to false. In this case, the current state's children have outdated probabilities; then we insert it into the set $\mathcal{S}$ to perform a downward traversal updating. (6) This graph traversal ends when $\mathcal{Q}$ becomes empty, i.e., no more nodes to update the reach or domain. Then, `UpdateDescendants` is performed to update the probabilities of the descendants of the nodes in $\mathcal{S}$.

**Algorithm 3.5** Update ancestor nodes of $d$

**procedure** UPDATEANCESTORS($c$, $p^*$, $\gamma$, $O$, $\mathcal{A}$)
  $\mathcal{S} \leftarrow \emptyset$
  Initialize a queue $\mathcal{Q}$
  $\mathcal{Q}.push(p^*, \textbf{true})$
  **while not** $\mathcal{Q}.empty()$ **do**
    $s, flag \leftarrow \mathcal{Q}.front()$
    $\mathcal{Q}.pop()$
    $change \leftarrow$ `UpdateState`$(c, s, O)$ ▷ Algorithm 3.6 in subsection 3.2.3
    $visited(s) \leftarrow$ **true**
    **if** $change \neq$ `none` **then**
      **if** $flag$ **and** $change =$ `reach` **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
        $flag \leftarrow$ **false**
      **end if**
      **for all** $p \in par(s)$ **do**
        **if not** $visited(p)$ **then**
          $\mathcal{Q}.push(p, flag)$
        **end if**
      **end for**
    **end if**
  **end while**
  `UpdateDescendants`$(\mathcal{S}, \gamma, O, \mathcal{A})$ ▷ Algorithm 3.7 in subsection 3.2.3
**end procedure**

The `UpdateState` is given in Algorithm 3.6 and follows the steps: (1) Verify whether for every node defined as reachable from $c$ is also defined as reachable from $s$, where $s$ is ancestor of $c$.

**Algorithm 3.6** Update State $s$

```
1: procedure UPDATESTATE(c, s, O)
2:     change ← none
3:     for all v ∈ V do
4:         if reach(c,v) and not reach(s,v) then
5:             reach(s,v) ← true
6:             if change ≠ domain then
7:                 change ← reach
8:             end if
9:             if ch(v) = ∅ then
10:                D_s ← D_s ∪ {A_v}
11:                σ_s ← σ_s + σ_v
12:                change ← domain
13:            end if
14:        end if
15:    end for
16:    return change
17: end procedure
```

(2) If this is not true for any node $v$, set $v$ as reachable from $s$. Further, if $v$ is a sink node, insert the column $A_v$ into the columns represented by $s$, $D_s$, and sum the topic vector $\sigma_v$ to $\sigma_s$. The `UpdateDescendants` operation is given in Algorithm 3.7 and performs the downward traversal update step. It applies a topological sort (Kahn, 1962) (`TopologicalSort` procedure) given the starting point nodes and the organization. This step retrieves a list of ordered states reachable from at least one of the given starting nodes so that we visit a state after all its ancestors within that set. More formally, it returns a list $Q = v_1, v_2, ..., v_n$, where if $v_i, v_j \in Q$ and $v_i v_j \in E$ then $i < j$. This approach allows updating the level and probabilities of the states according to the equation 3.4 in a consistent manner.

**Algorithm 3.7** Update descendant nodes of $p$

```
1: procedure UPDATEDESCENDANTS(P, γ, O, A)
2:     Q ← TopologicalSort(P, O)
3:     while not Q.empty() do
4:         s ← Q.front()
5:         Q.pop()
6:         visited(s) ← false
7:         level(s) ← min_{p∈par(s)} level(p) + 1
8:         for all X ∈ A do
9:             compute P(s|X, O) considering γ
10:        end for
11:    end while
12: end procedure
```

Finally, the `DeleteParent` operation performs the parent deletion, as presented in Algorithm 3.8, by the following steps: (1) Identify the parent $p^*$ of the chosen state $s$ with the most negligible reachability probability; (2) Then $p^*$ and its siblings are inserted into the set $\mathcal{P}$ of the states to be deleted; (3) For each state in $\mathcal{P}$, insert transitions from its parents to its children (4) Additionally, remove all state into $\mathcal{P}$ and their transitions from the graph; (5) Finally, the

`UpdateDescendants` function updates the probabilities starting from the parents of the deleted states.

**Algorithm 3.8** Deleting Parent Operation

**procedure** DELETEPARENT($O$, $l$, $s$, $\gamma$, $\mathcal{A}$)
  $\mathcal{P} \leftarrow \emptyset$
  $\mathcal{G} \leftarrow \emptyset$
  $p^* \leftarrow$ **null**
  **for all** $p \in par(s)$ **do**
    **if** $p^* =$ **null or** $P(p|O) < P(p^*|O)$ **then**
      $p^* \leftarrow p$
    **end if**
  **end for**
  **for all** $g \in par(p^*)$ **do**
    **for all** $s \in ch(g)$ **do**
      **if** $ch(s) \neq \emptyset$ **then**
        $\mathcal{P} \leftarrow \mathcal{P} \cup s$
      **end if**
    **end for**
  **end for**
  **for all** $p \in \mathcal{P}$ **do**
    **for all** $c \in ch(p)$ **do**
      $E \leftarrow E - \{pc\}$
      **for all** $g \in par(p)$ **do**
        $E \leftarrow E \cup \{gc\}$
      **end for**
    **end for**
    **for all** $g \in par(p)$ **do**
      $E \leftarrow E - \{gp\}$
      $\mathcal{G} \leftarrow \mathcal{G} \cup \{g\}$
    **end for**
  **end for**
  $\mathcal{O}' \leftarrow (V - \mathcal{P}, E)$
  $UpdateDescendants(\mathcal{G}, \gamma, O, \mathcal{A})$
  **return** $\mathcal{O}'$
**end procedure**

The `TopologicalSort` implementations usually have $O(|V| + |E|)$ time complexity. Since $E$ has at most $|V|^2$ edges in a DAG, $O(|V|^2)$ is also valid. Then, considering a digraph built by the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Murtagh and Contreras, 2011) and modified without adding new nodes, we have that $|V| \leq 2 \cdot |\mathcal{A}| - 1$; hence, $O(|\mathcal{A}|^2)$. For each attribute $X \in \mathcal{A}$ in the data lake, computing $P(s|X, O)$ requires iterating through all the parent nodes, which can be at most $|\mathcal{A}| - 1$. Then, it is in $O(|\mathcal{A}|^2)$. Consequently, because `UpdateDescendants` has the potential to traverse all nodes in the graph, its time complexity is $O(|\mathcal{A}|^3)$.

In `UpdateAncestors`, there is the possibility of visiting all $|\mathcal{A}| - 1$ non-sink nodes, with `UpdateState` being the most costly operation in each iteration. This function loops through all nodes of the graph (which is at most $2 \cdot |\mathcal{A}| - 1$), and for the sink nodes, it may sum vectors of $d$ dimensions, then its time complexity is $O(d \cdot |\mathcal{A}|)$. Thus, having `UpdateDescendants` executed at the end, `UpdateAncestors` has $O(|\mathcal{A}|^3 + d \cdot |\mathcal{A}|^2)$ time complexity. Assuming $d <$

$|\mathcal{A}|$, this can be simplified to $O(|\mathcal{A}|^3)$. Since $reach(\cdot,\cdot)$ and $P(\cdot|O)$ are $O(1)$ operations, we have that `UpdateAncestors` is the most costly operation in `AddParent`. Therefore, it is in $O(|\mathcal{A}|^3 + d \cdot |\mathcal{A}|^2)$. Analogous to `AddParent`, `UpdateDescendants` is the most costly operation in `DeleteParent`, so its time complexity is $O(|\mathcal{A}|^3)$.

### 3.2.4 Fast SA

A modified version of SA is also proposed in this work: Fast SA. This improvement uses dynamic programming to calculate transition probabilities. In the update probabilities step (`UpdateDescendants`' loop), we compute and save the following descriptor for the current $s \in Q$:

$$\mathcal{W}_s(X) = \sum_{t \in ch(s)} e^{\frac{\gamma}{|ch(s)|} \cdot \kappa(t,X)}. \tag{3.11}$$

Then, $\forall c \in ch(s)$, $c \in Q$ and

$$P(c|s,X,O) = \frac{e^{\frac{\gamma}{|ch(s)|} \cdot \kappa(c,X)}}{\mathcal{W}_s(X)}. \tag{3.12}$$

The topological sort ensures the consistency of this operation, since $\mathcal{W}_s(X)$ will be recomputed before $P(c|s,X,O)$. Another improvement is to compute $P(c|s,X,O)$ and $\mathcal{W}_c(X)$ if and only if $X \in D_c$. This strategy is based on the idea that most nodes can reach a small number of attributes. Its consistency lies in the fact that the effectiveness equation only involves discovery attribute probabilities $P(z_A|A,O)$. Consequently, each sink node $z_A \in V$ only requires parent probabilities related to its attribute $P(s|A,O)$, $\forall s \in par(z_A)$. Therefore, it is evaluated $P(s|A,O)$ $\forall A \in D_s$ if $ch(s)$ are sink nodes. Naturally, the concept extends to the nodes upwarding $s$.

As a result of this last optimization, it is no longer possible to evaluate the reachability probability defined as in the step (4) of Algorithm 3.3 description . Therefore, for the Fast SA, a new definition is given in the equation 3.13.

$$P(s|O) = \frac{1}{|D_s|} \sum_{A \in D_s} P(s|A,O) \tag{3.13}$$

This definition is better suited to its purpose in the optimization process, which is to identify states that need upgrades. However, since neighborhood structures consist of changing the parents of a node, it is preferable that this metric be directly related to the attribute discovery probabilities of its descendant sink nodes. Thus, an improvement in this metric necessarily implies an increase in one of these probabilities, unlike the old definition of reachability probability.

# 4

# Results and Discussion

The experimental results of the algorithms are evaluated according to the methodology presented in Barr et al. (1995). In other words, the analysis is carried out in terms of robustness, quality of the solutions, and computational efforts. This evaluation considers the same set of instances (test problems) and execution on the same machine.

We executed the experiments in an Intel i7-8565U ($1.80$ GHz and $4$ physical cores) computer with 16GB of RAM on an Ubuntu $18.04$ LTS operational system. To provide a comparative analysis of our proposition, we also implemented and executed *Organize* (Nargesian et al., 2023), the state-of-the-art algorithm, in the same environment. The C++ source code for such implementations is available for download (Fernandes et al., 2023b).

## 4.1 Benchmark Data Lakes

A fair comparison between our proposal and *Organize* would be achieved if they were evaluated in the same experimental environment used by Nargesian et al. (2020) in proposing their approach. However, the set of instances used in their experiments was not available. We propose benchmark data lakes to overcome this shortfall based on the methodology used to generate their instances. Like Nargesian et al. (2020), we collected datasets of various topics from the Socrata API (Socrata, 2022) and processed the metadata tags using their methodology, which will be discussed at the end of this subsection. Socrata is a popular open data lake that allows access to available data sets provided by government entities worldwide.

We provide $30$ benchmark data lakes partitioned into three groups according to their size. Instances can have $100$, $300$ or $500$ tables. In addition, there is no overlapping between instances from the same size group. Each instance has its tables uniformly distributed into a set of categories. A category is a topic of interest in society, represented by a label in the table's metadata. Each benchmark data lake is denoted as Socrata-M-N, where $M \in \{100, 300, 500\}$ indicates the

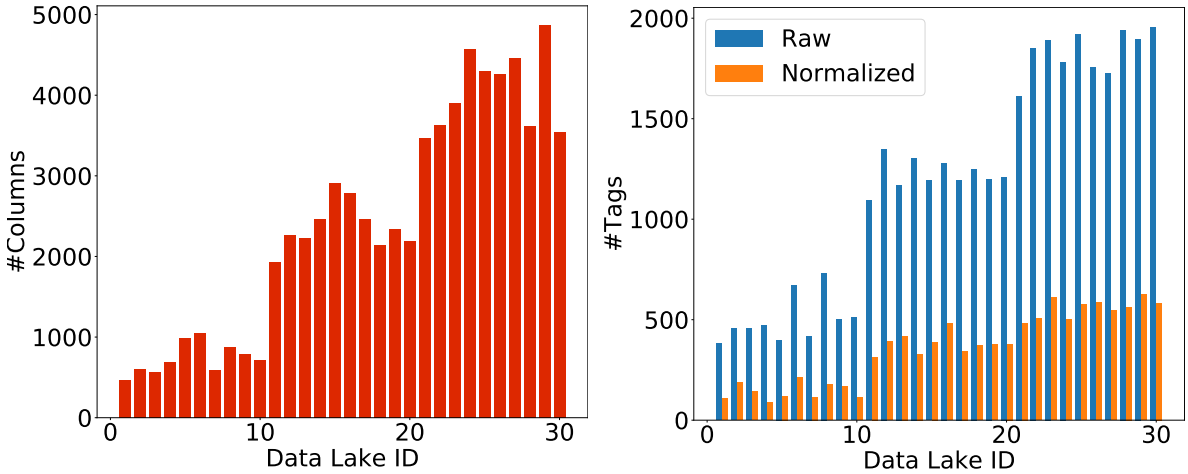
size and $N \in \{1, ..., 10\}$ indexes the set of categories. The chosen categories were:
A - Finance,
B - Social Services,
C - Demographics,
D - Infrastructure,
E - Health,
F - Environment,
G - Public Safety,
H - Education,
I - Economy,
J - Transportation.
Therefore, the sets of categories are indexed as follows:

Table 4.1: Sets of data categories

| Set | Categories |
|---|---|
| 1 | A |
| 2 | B, C |
| 3 | D, E, F |
| 4 | G – J |
| 5 | F – J |
| 6 | E – J |
| 7 | D – J |
| 8 | C – J |
| 9 | B – J |
| 10 | A – J |

This methodology provides data lake instances with different semantic contents and sizes, diversifying the difficulty and semantic complexity. Fig. 4.1(a) and 4.1(b) show the number of columns and tags per instance, respectively. For these figures, we ordered the instances on the x-axis primarily by the number of tables and secondarily by the number of categories. Tags are additional labels associated with tables provided by Socrata API. They differ in categories.

In Fig. 4.1(b), we see the count of the original tags and those obtained after normalization. This transformation reduces redundancy (Nargesian et al., 2020). Finally, in the pre-processing stage, all tables have their non-textual columns removed. The remaining columns, we transformed into topic vectors through a fastText model (Wang et al., 2020). All the benchmark data lakes — including their meta, raw, and vectorized data — are available for download (Fernandes et al., 2023b).




(a) Number of columns per benchmark data lake.

(b) Number of tags per benchmark data lake.

Figure 4.1: Summary of the benchmark data lakes.

## 4.2 Algorithms Tuning

We achieve a fair comparison between metaheuristics when both approaches have their parameters well-tuned and, at the same time, perform the search. We use `Irace` (López-Ibáñez et al., 2016) to automatically find the best parameters for the evaluated solutions.

The *Organize* algorithm for the DLOP has two parameters: maximum number of iterations without significant improvement ($k_f$) and relative improvement threshold ($\varepsilon$). They are used to identify a plateau and stop the search. The value range for the mentioned parameters is:

$$k_f \in [1, 150] \subset \mathbb{Z} \tag{4.1}$$

$$\varepsilon \in [0.01, 0.10] \subset \mathbb{R}. \tag{4.2}$$

For tuning our SA Algorithm, we define the parameters value range as follows:

$$\alpha \in [0.001, 0.010] \subset \mathbb{R} \tag{4.3}$$

$$k_i \in [1, 100] \subset \mathbb{Z} \tag{4.4}$$

$$k_f \in [1, 50] \subset \mathbb{Z} \tag{4.5}$$

We also set $\gamma = 25.0$ and the maximum elapsed time to $100$s for both metaheuristics. When an algorithm finishes before this threshold, it restarts and searches for another solution, keeping the best. The tuning process also requires a set of instances, so we create $10$ more data instances with $500$ tables each. We use these instances exclusively to calibrate the algorithms and present different tables from those previously discussed. For such input, `Irace` has recommended for *Organize* and our approach the parameters shown in Table 4.2. Since Fast SA is a variant of SA with a smaller number of computations and a more appropriate metric for node selection, we considered the same parameters for both.

Table 4.2: Recommend parameters by `Irace` for *Organize* and our proposal (SA)

| **Organize (O)** | | **Our prop. (SA)** | | |
|---|---|---|---|---|
| $\varepsilon$ | $k_f$ | $\alpha$ | $k_i$ | $k_f$ |
| 0.019 | 79 | 0.0094 | 1 | 31 |

## 4.3 Performance Evaluation

We conduct the performance evaluation for *Organize* and our SA Algorithms over the benchmark data lakes described previously and using the parameters recommended by `Irace`. Additionally, we assumed $\gamma = 25.0$ and limited the maximum elapsed time to $300$ seconds. Again, when a heuristic finishes before that threshold, it restarts and searches for another solution, keeping the best one.

### 4.3.1 Effectiveness

To evaluate the effectiveness of all approaches, we execute them $15$ times for each benchmark data lake. Tables 4.3 – 4.5 provide the fitness of the best solution ($E_{best}$) achieved by three heuristics and the average number of heuristic restarts ($\hat{\mu}_r$) for each instance. In addition, we provide the relative percentage improvement

$$Imp\% = \frac{E^{SA}_{best} - E^{O}_{best}}{E^{SA}_{best}} \times 100, \tag{4.6}$$

SA is one of our solutions, and O is the *Organize* one. This metric quantifies the increase in maximum effectiveness reached by our proposal ($E^{SA}_{best}$) compared to the *Organize* ($E^{O}_{best}$).

The greatest $E_{best}$ for each instance is bold. Our proposals outperformed *Organize* in all instances, with Fast SA always outperforming the other two heuristics. The top three $Imp\%$ in each size group are underlined. The best result was in Socrata-500-6, where the $Imp\%$ was $44.54\%$. The average $Imp\%$ for SA was $9.21\%$, $4.95\%$ and $3.97\%$ for the size groups $100$, $300$ and $500$, respectively. As for Fast SA, they were $13.27\%$, $14.08\%$ and $22.78\%$ for the respective groups. On average, our approaches performed more restarts than *Organize*, which suggests that the former converges faster.

We execute a sign test on the $E_{best}$ pairs (Taillard et al., 2008) to provide statistical evidence of the Fast SA's superiority. For this, we applied the test to the pairs *Organize*-SA, *Organize*-Fast SA and SA-FastSA. The null hypothesis assumes that there is no superiority of one approach over the other. In other words, the probability of heuristic $X$ obtaining a better solution than $Y$ is $0.5$, i.e. $P(E^{X}_{best} > E^{Y}_{best}) = P(E^{Y}_{best} > E^{X}_{best}) = 0.5$. Thus, given $n = 30$ trials in different instances, the number of occurrences $K$ in which $E^{X}_{best} > E^{Y}_{best}$ is a a random variable that obeys a binomial distribution, i.e. $K = \#(E^{X}_{best} > E^{Y}_{best}) \sim bin(n = 30, p = 0.5)$.

Table 4.3: Best solution effectiveness and average number of restarts achieved in organizing benchmark data lakes with 100 tables

| Instance | Organize (O) | | Our prop. (SA) | | | Our prop. (Fast SA) | | |
|---|---|---|---|---|---|---|---|---|
| | $E_{best}$ | $\hat{\mu}_r$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ |
| Socrata-100-1 | 0.3056 | 45.1 | 0.3392 | 256.9 | 10.99 | **0.3545** | 722.1 | 16.00 |
| Socrata-100-2 | 0.3388 | 27.1 | 0.3813 | 179.9 | 12.54 | **0.3892** | 456.5 | 14.87 |
| Socrata-100-3 | 0.3713 | 29.8 | 0.4012 | 182.8 | 8.05 | **0.4067** | 582.2 | 9.53 |
| Socrata-100-4 | 0.3253 | 19.8 | 0.3638 | 123.9 | 11.84 | **0.3771** | 366.4 | 15.92 |
| Socrata-100-5 | 0.4710 | 10.3 | 0.4824 | 69.2 | 2.42 | **0.5015** | 169.4 | 6.48 |
| Socrata-100-6 | 0.3638 | 9.6 | 0.3897 | 60.6 | 7.12 | **0.4081** | 129.6 | 12.18 |
| Socrata-100-7 | 0.3718 | 26.2 | 0.4306 | 175.4 | 15.81 | **0.4408** | 413.73 | 18.56 |
| Socrata-100-8 | 0.2918 | 13.1 | 0.3298 | 86.0 | 13.02 | **0.3341** | 154.27 | 14.50 |
| Socrata-100-9 | 0.3763 | 15.3 | 0.3884 | 106.1 | 3.22 | **0.4310** | 283.73 | 14.54 |
| Socrata-100-10 | 0.3799 | 15.5 | 0.4067 | 120.7 | 7.05 | **0.4182** | 337.6 | 10.08 |
| **Average** | | | | | **9.21** | | | **13.27** |

Table 4.4: Best solution effectiveness and average number of restarts achieved in organizing benchmark data lakes with 300 tables.

| Instance | Organize (O) | | Our prop. (SA) | | | Our prop. (Fast SA) | | |
|---|---|---|---|---|---|---|---|---|
| | $E_{best}$ | $\hat{\mu}_r$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ |
| Socrata-300-1 | 0.1795 | 3.0 | 0.2011 | 17.8 | 12.03 | **0.2312** | 35.9 | 28.80 |
| Socrata-300-2 | 0.2281 | 2.0 | 0.2281 | 13.0 | 0.0 | **0.2301** | 34.7 | 0.88 |
| Socrata-300-3 | 0.1698 | 2.0 | 0.1699 | 13.3 | 0.06 | **0.2017** | 29.3 | 18.79 |
| Socrata-300-4 | 0.2400 | 2.0 | 0.2409 | 11.0 | 0.38 | **0.2485** | 20.5 | 3.54 |
| Socrata-300-5 | 0.1643 | 2.0 | 0.1719 | 8.0 | 4.63 | **0.1953** | 13.3 | 18.87 |
| Socrata-300-6 | 0.1633 | 2.0 | 0.1851 | 8.3 | 13.35 | **0.1866** | 12.5 | 14.27 |
| Socrata-300-7 | 0.2100 | 2.0 | 0.2170 | 11.0 | 3.33 | **0.2303** | 20.67 | 9.67 |
| Socrata-300-8 | 0.1866 | 2.0 | 0.1953 | 14.3 | 4.66 | **0.2003** | 29.5 | 7.34 |
| Socrata-300-9 | 0.1983 | 2.0 | 0.2109 | 12.1 | 6.35 | **0.2292** | 23.87 | 15.58 |
| Socrata-300-10 | 0.1864 | 2.0 | 0.1951 | 14.2 | 4.67 | **0.2312** | 29.93 | 23.03 |
| **Average** | | | | | **4.95** | | | **14.08** |

The alternative hypothesis is that $p > 0.5$ and the $p$-value is $P(K \geq k)$, where $k$ is the number of $E^X_{best} > E^Y_{best}$ events observed in the experiment. The hypotheses of the tests and their p-values are shown in the Table 4.6. As a result, we get a $p$-value$= 9.3 \cdot 10^{-10}$ in all tests, which leads to rejecting the null hypothesis and assuming the alternative one at $0.01$ significance level. Therefore, our proposal has superior effectiveness and Fast SA is the best one.

Tables 4.7 – 4.9 provide the sample mean, standard deviation fitness of both solutions. Each instance's most significant $\hat{\mu}_E$ is bold. Note that in terms of average, our solution Fast SA still holds better outcomes. We applied the Mann-Whitney test (Mann and Whitney, 1947), considering all fitness values of each pair of solutions to inspect the improvement of our solutions in

each data lake. We can use this non-parametric test when the data's normality assumption is not satisfied, as in heuristic performance data (Ribeiro et al., 2011).

The null hypothesis states that the median effectiveness of the solutions generated by heuristics $X$ and $Y$ are equal, i.e, $m_E^X = m_E^Y$. The alternative presumes that $m_E^X > m_E^Y$. Tables 4.10 – 4.12 provide the hypothesis for each pair of heuristics with the p-values obtained for each data lake through the test. The greatest p-value in each table is in bold. At the $0.01$ level, we discard the null hypothesis and keep the one stating the prevalence of our solutions. Furthermore, we infer again that Fast SA was dominant.

Table 4.5: Best solution effectiveness and average number of restarts achieved in organizing benchmark data lakes with $500$ tables.

| **Instance** | **Organize (O)** | | **Our prop. (SA)** | | | **Our prop. (Fast SA)** | | |
|---|---|---|---|---|---|---|---|---|
| | $E_{best}$ | $\hat{\mu}_r$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ | $E_{best}$ | $\hat{\mu}_r$ | $Imp\%$ |
| Socrata-500-1 | 0.1240 | 1.0 | 0.1242 | 5.3 | 0.16 | **0.1479** | 11.93 | 19.27 |
| Socrata-500-2 | 0.1228 | 1.0 | 0.1235 | 5.0 | 0.57 | **0.1600** | 10.53 | 30.29 |
| Socrata-500-3 | 0.1126 | 1.0 | 0.1128 | 4.8 | 0.18 | **0.1340** | 7.67 | 19.01 |
| Socrata-500-4 | 0.1283 | 1.0 | 0.1285 | 3.3 | 0.16 | **0.1479** | 5.2 | 15.28 |
| Socrata-500-5 | 0.1275 | 1.0 | 0.1338 | 4.0 | 4.94 | **0.1597** | 6.07 | 25.25 |
| Socrata-500-6 | 0.1089 | 1.0 | 0.1248 | 4.0 | 14.60 | **0.1574** | 6.67 | 44.54 |
| Socrata-500-7 | 0.1266 | 1.0 | 0.1274 | 3.5 | 0.63 | **0.1491** | 6.27 | 17.77 |
| Socrata-500-8 | 0.1090 | 1.0 | 0.1171 | 5.0 | 7.43 | **0.1203** | 9.27 | 10.37 |
| Socrata-500-9 | 0.1013 | 1.0 | 0.1023 | 3.0 | 0.99 | **0.1161** | 4.53 | 14.61 |
| Socrata-500-10 | 0.1372 | 1.0 | 0.1510 | 5.3 | 10.06 | **0.1803** | 11.2 | 31.41 |
| **Average** | | | | | **3.97** | | | **22.78** |

Table 4.6: Sign test conducted to assess the dominance of one heuristic over another, pairwise analyzing the best effectiveness achieved over all the benchmark data lakes.

| $H_0 : P(E_{best}^{SA} > E_{best}^{O}) = 0.5$ | $H_0 : P(E_{best}^{FastSA} > E_{best}^{O}) = 0.5$ | $H_0 : P(E_{best}^{FastSA} > E_{best}^{SA}) = 0.5$ |
|---|---|---|
| $H_a : P(E_{best}^{SA} > E_{best}^{O}) > 0.5$ | $H_a : P(E_{best}^{FastSA} > E_{best}^{O}) > 0.5$ | $H_a : P(E_{best}^{FastSA} > E_{best}^{SA}) > 0.5$ |
| $9.31 \cdot 10^{-10}$ | $9.31 \cdot 10^{-10}$ | $9.31 \cdot 10^{-10}$ |

### 4.3.2 Efficiency

We perform another experiment with a graphical analysis through time-to-target (ttt) plots (Aiex et al., 2006) to conduct the efficiency evaluation. Such a plot shows the empirical cumulative distribution function of the time required for an algorithm to produce a solution and a target. For this analysis, we choose the problem instances Socrata-$100$-$3$, Socrata-$100$-$5$, Socrata-$300$-$6$, and Socrata-$500$-$6$. We chose the first two because they are the ones in which *Organize*

Table 4.7: Effectiveness sample mean and standard deviation achieved in organizing benchmark data lakes with 100 tables.

| Instance | Organize (O) | | Our prop. (SA) | | Our prop. (Fast SA) | |
|---|---|---|---|---|---|---|
| | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ |
| Socrata-100-1 | 0.3056 | 0.0000 | 0.3336 | 0.0039 | **0.3485** | 0.0039 |
| Socrata-100-2 | 0.3388 | 0.0000 | 0.3649 | 0.0049 | **0.3825** | 0.0025 |
| Socrata-100-3 | 0.3615 | 0.0109 | 0.4010 | 0.0001 | **0.4056** | 0.0010 |
| Socrata-100-4 | 0.3253 | 0.0000 | 0.3574 | 0.0126 | **0.3718** | 0.0048 |
| Socrata-100-5 | 0.4576 | 0.0084 | 0.4817 | 0.0006 | **0.4891** | 0.0036 |
| Socrata-100-6 | 0.3638 | 0.0000 | 0.3873 | 0.0047 | **0.4022** | 0.0063 |
| Socrata-100-7 | 0.3718 | 0.0000 | 0.4046 | 0.0073 | **0.4356** | 0.0039 |
| Socrata-100-8 | 0.2918 | 0.0000 | 0.3190 | 0.0050 | **0.3256** | 0.0049 |
| Socrata-100-9 | 0.3763 | 0.0000 | 0.3845 | 0.0017 | **0.4178** | 0.0073 |
| Socrata-100-10 | 0.3799 | 0.0000 | 0.4028 | 0.0088 | **0.4124** | 0.0052 |

presents more variability in its solutions' quality. We chose the last two because, in them, we scored the best in the size groups 300 and 500, respectively.

Table 4.8: Effectiveness sample mean and standard deviation achieved in organizing benchmark data lakes with 300 tables.

| Instance | Organize (O) | | Our prop. (SA) | | Our prop. (Fast SA) | |
|---|---|---|---|---|---|---|
| | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ |
| Socrata-300-1 | 0.1795 | 0.0000 | 0.1831 | 0.0068 | **0.2115** | 0.0125 |
| Socrata-300-2 | 0.2281 | 0.0000 | 0.2281 | 0.0000 | **0.2298** | 0.0006 |
| Socrata-300-3 | 0.1698 | 0.0000 | 0.1699 | 0.0000 | **0.1859** | 0.0095 |
| Socrata-300-4 | 0.2400 | 0.0000 | 0.2402 | 0.0003 | **0.2441** | 0.0036 |
| Socrata-300-5 | 0.1643 | 0.0000 | 0.1650 | 0.0019 | **0.1838** | 0.0087 |
| Socrata-300-6 | 0.1633 | 0.0000 | 0.1684 | 0.0082 | **0.1804** | 0.0062 |
| Socrata-300-7 | 0.2100 | 0.0000 | 0.2120 | 0.0023 | **0.2181** | 0.0040 |
| Socrata-300-8 | 0.1866 | 0.0000 | 0.1882 | 0.0021 | **0.1960** | 0.0039 |
| Socrata-300-9 | 0.1983 | 0.0000 | 0.2013 | 0.0035 | **0.2220** | 0.0102 |
| Socrata-300-10 | 0.1864 | 0.0000 | 0.1872 | 0.0022 | **0.2232** | 0.0119 |

We measure the CPU time (in seconds) for each problem instance to find an objective function value at least as good as a given target value. Each heuristic is run 50 times on the fixed instance and using the given target solution. We define the target value for each instance as the sample average of the fitness obtained by *Organize*. Such values were 0.3615, 0.4576, 0.1633, and 0.1089 for the instances Socrata-100-3, Socrata-100-5, Socrata-300-6, and Socrata-500-6, respectively. However, we rounded those values to four decimal digits, limiting the elapsed time to 300 seconds.

Table 4.9: Effectiveness sample mean and standard deviation achieved in organizing benchmark data lakes with 500 tables.

| Instance | Organize (O) | | Our prop. (SA) | | Our prop. (Fast SA) | |
|---|---|---|---|---|---|---|
| | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ | $\hat{\mu}_E$ | $\hat{\sigma}_E$ |
| Socrata-500-1 | 0.1240 | 0.0000 | 0.1241 | 0.0000 | **0.1342** | 0.0099 |
| Socrata-500-2 | 0.1228 | 0.0000 | 0.1229 | 0.0002 | **0.1415** | 0.0115 |
| Socrata-500-3 | 0.1126 | 0.0000 | 0.1127 | 0.0001 | **0.1206** | 0.0101 |
| Socrata-500-4 | 0.1283 | 0.0000 | 0.1284 | 0.0001 | **0.1334** | 0.0078 |
| Socrata-500-5 | 0.1275 | 0.0000 | 0.1281 | 0.0016 | **0.1372** | 0.0115 |
| Socrata-500-6 | 0.1089 | 0.0000 | 0.1109 | 0.0049 | **0.1267** | 0.0121 |
| Socrata-500-7 | 0.1266 | 0.0000 | 0.1268 | 0.0003 | **0.1362** | 0.0095 |
| Socrata-500-8 | 0.1090 | 0.0000 | 0.1107 | 0.0025 | **0.1142** | 0.0034 |
| Socrata-500-9 | 0.1013 | 0.0000 | 0.1015 | 0.0003 | **0.1053** | 0.0047 |
| Socrata-500-10 | 0.1372 | 0.0000 | 0.1382 | 0.0035 | **0.1691** | 0.0115 |

Fig. 4.2(a) shows the time to target for the SA, Fast SA and *Organize* in the problem instance Socrata-100-3. The superiority of our approaches is straightforward. In these, the probability of reaching the target at $t = 25s$ is $1.0$. However, in *Organize*, the probability for this case is less than $0.3$. At $t = 4s$ such a probability is preserved for Fast SA, but for SA it drops to $0.42$ and *Organize* to $0.03$. In addition, the probability of the last one finds a solution as good as the target at most $300s$ is almost $0.6$. That is, there is a probability greater than $0.4$ that the literature algorithm will reach the time limit without hitting the target.

Table 4.10: Mann-Whitney test conduced to compare the median effectiveness achieved by each heuristic on each benchmark data lake with 100 tables.

| Instance | $H_0 : m_E^{SA} = m_E^O$<br>$H_a : m_E^{SA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^O$<br>$H_a : m_E^{FastSA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^{SA}$<br>$H_a : m_E^{FastSA} > m_E^{SA}$ |
|---|---|---|---|
| Socrata-100-1 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |
| Socrata-100-2 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $2.0 \cdot 10^{-6}$ |
| Socrata-100-3 | $1.1 \cdot 10^{-6}$ | $1.2 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |
| Socrata-100-4 | $3.3 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $4.3 \cdot 10^{-6}$ |
| Socrata-100-5 | $9.2 \cdot 10^{-7}$ | $9.1 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |
| Socrata-100-6 | $3.3 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |
| Socrata-100-7 | $3.3 \cdot 10^{-7}$ | $3.3 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |
| Socrata-100-8 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $\mathbf{2.6 \cdot 10^{-4}}$ |
| Socrata-100-9 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.7 \cdot 10^{-6}$ |
| Socrata-100-10 | $2.6 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |

The Socrata-100-5 instance is more complex in the time to target of Fig. 4.2(b). Indeed, our SA and Fast SA reach the target at most $t = 9s$ with a probability around $0.35$ and $1.0$,

Table 4.11: Mann-Whitney test conduced to compare the median effectiveness achieved by each heuristic on each benchmark data lake with 300 tables.

| **Instance** | $H_0 : m_E^{SA} = m_E^O$<br>$H_a : m_E^{SA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^O$<br>$H_a : m_E^{FastSA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^{SA}$<br>$H_a : m_E^{FastSA} > m_E^{SA}$ |
|---|---|---|---|
| Socrata-300-1 | $3.4 \cdot 10^{-7}$ | $1.5 \cdot 10^{-6}$ | $1.2 \cdot 10^{-6}$ |
| Socrata-300-2 | $1.4 \cdot 10^{-5}$ | $1.5 \cdot 10^{-6}$ | $7.5 \cdot 10^{-5}$ |
| Socrata-300-3 | $3.4 \cdot 10^{-7}$ | $1.5 \cdot 10^{-6}$ | $1.7 \cdot 10^{-5}$ |
| Socrata-300-4 | $3.4 \cdot 10^{-7}$ | $1.5 \cdot 10^{-6}$ | $1.4 \cdot 10^{-4}$ |
| Socrata-300-5 | $1.3 \cdot 10^{-6}$ | $1.5 \cdot 10^{-6}$ | $1.8 \cdot 10^{-5}$ |
| Socrata-300-6 | $3.4 \cdot 10^{-7}$ | $1.5 \cdot 10^{-6}$ | $\mathbf{1.5 \cdot 10^{-3}}$ |
| Socrata-300-7 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $8.0 \cdot 10^{-6}$ |
| Socrata-300-8 | $3.4 \cdot 10^{-7}$ | $4.5 \cdot 10^{-7}$ | $2.4 \cdot 10^{-7}$ |
| Socrata-300-9 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.4 \cdot 10^{-5}$ |
| Socrata-300-10 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $1.6 \cdot 10^{-6}$ |

respectively. In contrast, the likelihood of the *Organize* reaching the goal in at most $9s$ is $0.03$. In addition, there is a probability greater than $0.7$ that the algorithm will get the time limit without hitting the target. For the Socrata-300-6 and Socrata-500-6, whose time to target (Figs. 4.2(c) and 4.2(d)), the *Organize* algorithm failed to reach the goal within the time limit.

Table 4.12: Mann-Whitney test conduced to compare the median effectiveness achieved by each heuristic on each benchmark data lake with 500 tables.

| **Instance** | $H_0 : m_E^{SA} = m_E^O$<br>$H_a : m_E^{SA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^O$<br>$H_a : m_E^{FastSA} > m_E^O$ | $H_0 : m_E^{FastSA} = m_E^{SA}$<br>$H_a : m_E^{FastSA} > m_E^{SA}$ |
|---|---|---|---|
| Socrata-500-1 | $4.3 \cdot 10^{-6}$ | $3.4 \cdot 10^{-7}$ | $1.7 \cdot 10^{-6}$ |
| Socrata-500-2 | $1.0 \cdot 10^{-4}$ | $3.4 \cdot 10^{-7}$ | $1.4 \cdot 10^{-6}$ |
| Socrata-500-3 | $4.1 \cdot 10^{-5}$ | $3.4 \cdot 10^{-7}$ | $1.3 \cdot 10^{-3}$ |
| Socrata-500-4 | $1.4 \cdot 10^{-5}$ | $3.4 \cdot 10^{-7}$ | $\mathbf{3.5 \cdot 10^{-3}}$ |
| Socrata-500-5 | $1.3 \cdot 10^{-6}$ | $3.4 \cdot 10^{-7}$ | $4.0 \cdot 10^{-5}$ |
| Socrata-500-6 | $4.3 \cdot 10^{-6}$ | $3.4 \cdot 10^{-7}$ | $4.8 \cdot 10^{-5}$ |
| Socrata-500-7 | $4.1 \cdot 10^{-5}$ | $3.4 \cdot 10^{-7}$ | $2.5 \cdot 10^{-6}$ |
| Socrata-500-8 | $3.4 \cdot 10^{-7}$ | $3.4 \cdot 10^{-7}$ | $2.0 \cdot 10^{-3}$ |
| Socrata-500-9 | $4.1 \cdot 10^{-5}$ | $3.4 \cdot 10^{-7}$ | $6.6 \cdot 10^{-5}$ |
| Socrata-500-10 | $1.4 \cdot 10^{-5}$ | $3.4 \cdot 10^{-7}$ | $2.5 \cdot 10^{-6}$ |

In contrast, our Fast SA always reaches the goal in less than $32s$ and $135s$, respectively. For these instances and cut-off times, the SA hits the target with $0.7$ probability. The sudden jump observed near the maximum time to target in Figure 4.2 for the *Organize* algorithm provides valuable insights into its performance dynamics within the context of data lake organization. This behavior can be attributed to the algorithm's inherent difficulty in efficiently reaching the target

value within shorter time frames, highlighting its diminishing returns in effectiveness beyond the fourth decimal place.

Notably, this phenomenon becomes more pronounced in larger data lake environments, where the complexity of the dataset exacerbates the algorithm's limitations. The challenge then emerges in the pursuit of further improvements in initial data organization, as reducing the target value to a level that triggers a different behavior would necessitate grappling with a myriad of decimal places.

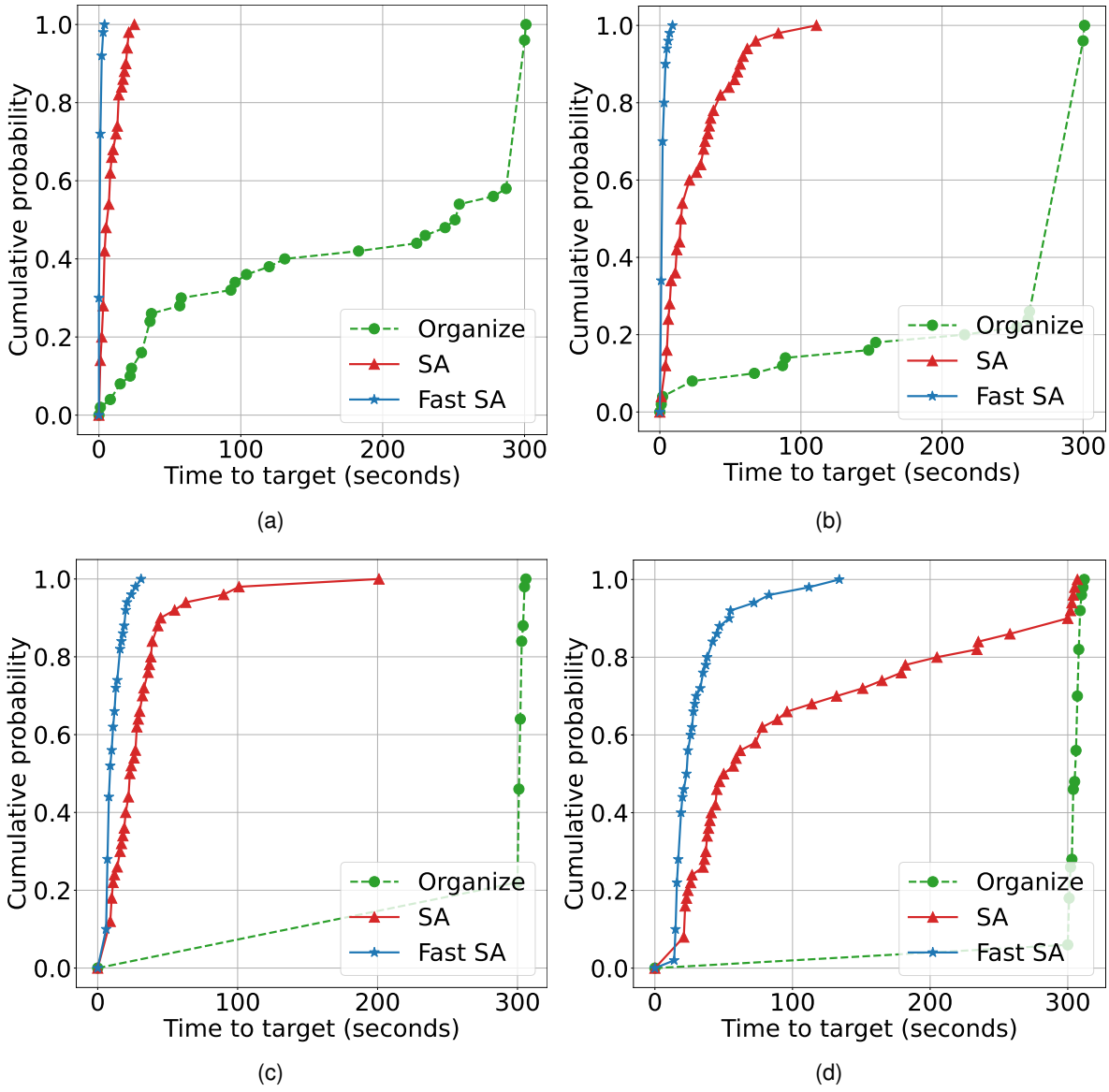



Figure 4.2: Time to target for instances (a) Socrata-100-3; (b) Socrata-100-5; (c) Socrata-300-6; (d) Socrata-500-6.

### 4.3.3 Success Probability

For additional evaluation, we simulate a user by the success probability of finding each table in the data lake. Such a measure considers navigation successful if it finds tables with the exact or a similar attribute to the query's attribute (Nargesian et al., 2023). Let $\kappa$ be a similarity measure between two attributes and $0 < \theta \leq 1$ be a threshold. The success probability of an attribute $A$ is defined as (Nargesian et al., 2023)

$$Success(A|O) = 1 - \prod_{A_i \in \mathcal{A} \wedge \kappa(A_i, A) \geq \theta} (1 - P(A_i|O)) \tag{4.7}$$

We use the cosine similarity on the topic vectors of attributes for $\kappa$ and $\theta = 0.9$. From the attribute success probabilities, we can compute table success probability as $Success(T|O) = 1 - \prod_{A \in T}(1 - Success(A|O))$ (Nargesian et al., 2023).

Figs. 4.3(b) – 4.3(d) show the table success probabilities from different approaches in organizing Socrata-100-1, Socrata-100-7, Socrata-300-1, and Socrata-500-6, respectively. We sorted the tables from lowest to highest probability on the x-axis. We also include two baselines to reinforce the organization's improvement in user navigation. The first considers no organizational structure, modeled as a flat organization where we connect all sink nodes directly to the source node (Nargesian et al., 2023).

Data labeling is a common and straightforward way to organize a data lake (Sawadogo and Darmont, 2021). We consider our labeling as proposed by Nargesian et al. (2023). They induce an organizational structure by labeled tables, where each unique label is to an internal node connected to their respective columns represented by sink nodes. Finally, They connect all internal nodes to the source node. Such a method is the second baseline. To eliminate redundancy, we merge labels provided by metadata through syntactic, semantic, and structural normalization (Fernandes et al., 2023b).

The Socrata API metadata provides some sets of labels. The used in this step was *domain tags*, different from *category* used previously in the data collection. The reason was to avoid the bias inherent in creating instances and enriching them with more metadata. For example, for instances built from a single category, the organization inferred from the metadata would be flat. Domain tags bring each dataset a set of tags with greater detail than the category. In this way, generating a more effective organization than that achieved by categories is possible because it provides a better distinction between datasets.

In Figs. 4.3(a) – 4.3(d), the first baseline, the second one, the *Organize* approach, and our proposals are denoted respectively by No Org, Metadata, Organize, SA and Fast SA. As expected, the flat baseline was the least successful structure for user navigation, and the metadata tag-induced approach provides a considerable improvement over that. The plots strongly indicate that the algorithms for DLOP provide a more successful navigation structure than the baselines. Additionally, our proposals proved more successful than the *Organize* one algorithm.

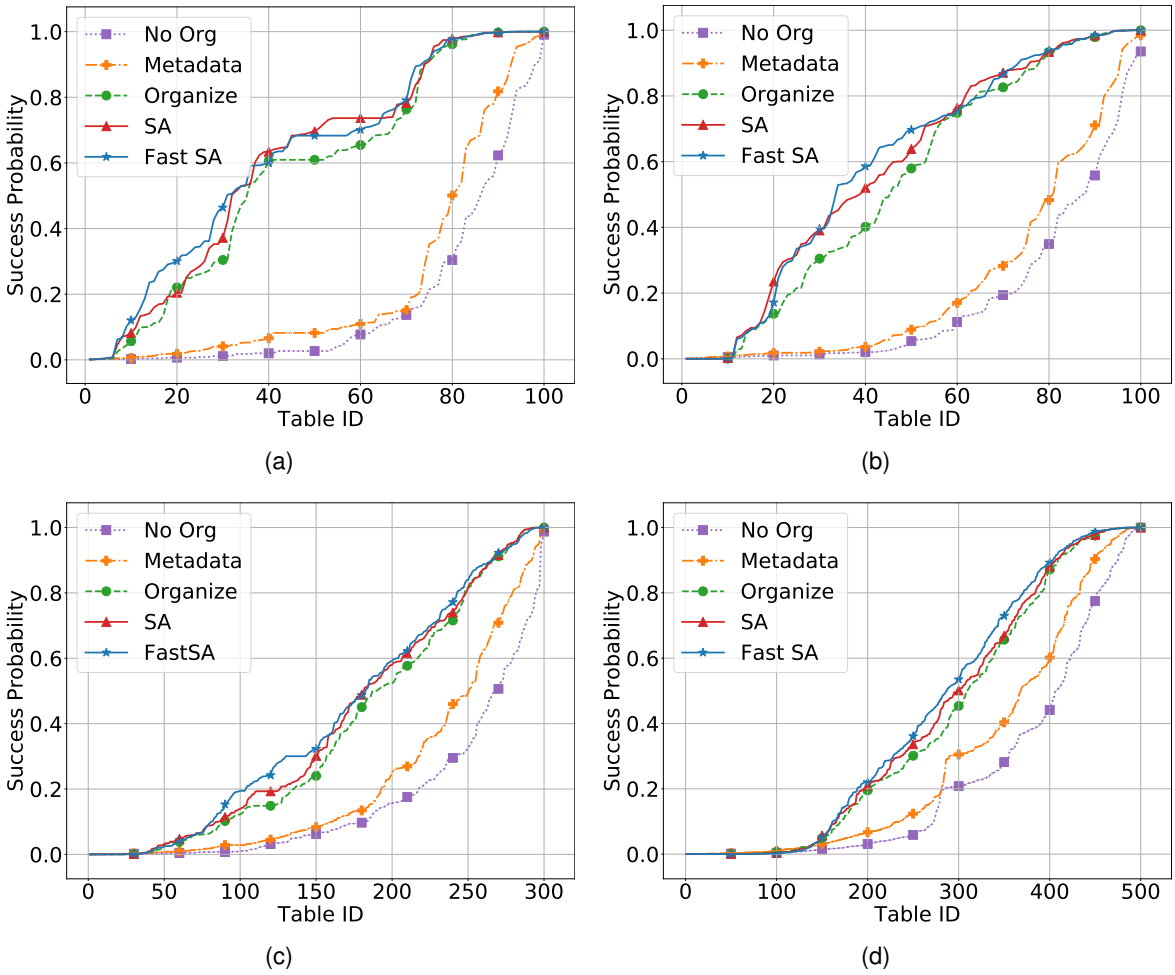



Figure 4.3: Tables success probabilities per organization approach for (a) Socrata-100-1; (b) Socrata-100-7; (c) Socrata-300-1; (d) Socrata-500-6.

The mean difference between their success probabilities are $0.036$, $0.044$, $0.021$ and $0.015$ for SA in Socrata-100-1, Socrata-100-7, Socrata-300-1 and Socrata-500-6, respectively. For Fast SA, these indicators increase for $0.048$, $0.052$, $0.038$ and $0.032$, respectively. Such analysis corroborates the results discussed earlier for the superiority of our proposals in producing more effective organizations.

# 5

# Final Considerations

This work presented two new approaches for generating navigation structures for data lakes. Our approaches consist of Multi-Start Simulated Annealing variants for the DLOP. Such metaheuristics generates random neighbor solutions through a roulette wheel selection-based approach. We propose algorithms for updating an organization after edge addition and node deletion. To compare with the state-of-the-art, called here *Organize*, we propose a set of 30 benchmark data lakes with different levels of complexity.

We applied the approaches for each instance and saved their best solutions. In our best result, our proposal outperforms the *Organize* by generating a $44.5\%$ more effective solution. To compare the efficiency, we evaluate the time to target that reveals our approaches' faster convergence. Further, we compute the success probability of finding tables for the approaches and two baselines. Again, our proposals was the most successful.

Our methodology addresses a critical gap in the literature by algorithmically describing these operations and ensuring the consistency of probabilities throughout the graph. Furthermore, we demonstrate the efficiency of our implementation, showcasing its potential to streamline the process of organizing data within data lakes. However, it's important to note some limitations. While our approach offers promising advantages, it may face challenges in specific scenarios, such as highly complex data lake architectures or massive datasets.

Additionally, the computational complexity associated with implementing our methodology may require careful consideration of computational resources. Finally, the main limitation of our work is that it only deals with textual tabular data and does not consider the real scenario of data distributed in computer clusters. Despite these potential drawbacks, our novel strategy represents a significant step forward in data lake management. It offers theoretical advancements and practical benefits for organizations grappling with the complexities of managing diverse datasets within their data lakes.

In future works, we plan to implement the organizing algorithms in CUDA for GPU parallel processing. Also, we plan to employ other metaheuristics to the organization problem and

develop new neighborhood structures. A further idea is the generalization of the Data Lake Organization Problem for unstructured data. Finally, adapting the MapReduce paradigm and implementing it on Apache Spark is also an interesting issue.

In terms of academic papers, part of the findings presented in this text have been submitted to the journal Soft Applied Computing and are currently under review. Nevertheless, the following works were already published during the master's program: • **Toward Data Lake Technologies for Intelligent Societies and Cities** (Ramos et al., 2023): In this work, we provided an overview of data lake technologies and recent literature contributions while also discussing its relevance in the context of sustainable smart cities. Finally, we proposed a data lake architecture improved with a robust metadata management system composed of the most promising techniques in the literature. • **Towards Edge-Based Data Lake Architecture for Intelligent Transportation System** (Fernandes et al., 2023a): This paper proposed an Edge-based Data Lake Architecture to integrate and analyze the complex data from ITS efficiently. The architecture offers scalability, fault tolerance, and performance, improving decision-making and enhancing innovative services for a more intelligent transportation ecosystem.

# References


*Handbook of Metaheuristics*. Springer US, 2010. ISBN 9781441916655. **DOI** 10.1007/978-1-4419-1665-5. URL http://dx.doi.org/10.1007/978-1-4419-1665-5.

Renata M. Aiex, Mauricio G. C. Resende, and Celso C. Ribeiro. TTT plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, October 2006.

A. A. Alekseev, V. V. Osipova, M. A. Ivanov, A. Klimentov, N. V. Grigorieva, and H. S. Nalamwar. Efficient data management tools for the heterogeneous big data warehouse. *Physics of Particles and Nuclei Letters*, 13(5):689–692, September 2016.

Syed Muhammad Fawad Ali and Robert Wrembel. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *The VLDB Journal*, 26(6):777–801, September 2017.

Taro Aso, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Relation-oriented faceted search method for knowledge bases. In *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services*, page 192–199. Association for Computing Machinery, 2020.

Mohamed Bakli, Mahmoud Sakr, and Esteban Zimanyi. Distributed Moving Object Data Management in MobilityDB. In *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. Association for Computing Machinery, 2019.

Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G. C. Resende, and William R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, September 1995. ISSN 1572-9397. **DOI** 10.1007/bf02430363. URL http://dx.doi.org/10.1007/BF02430363.

Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. Discovering Related Data At Scale. *Proceedings of the VLDB Endowment*, 14(8):1392–1400, april 2021.

Simon Elias Bibri. The IoT for smart sustainable cities of the future: An analytical framework for sensor-based big data applications for environmental sustainability. *Sustainable Cities and Society*, 38:230–253, 2018.

Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, page 1365–1375. Association for Computing Machinery, 2019.

Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

Sara Ceschia and Andrea Schaerf. Multi-neighborhood simulated annealing for the capacitated facility location problem with customer incompatibilities. *Computers amp; Industrial Engineering*, 188:109858, February 2024. ISSN 0360-8352. **DOI** 10.1016/j.cie.2023.109858. URL http://dx.doi.org/10.1016/j.cie.2023.109858.

Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. Multi-neighborhood simulated annealing for the minimum interference frequency assignment problem. *EURO Journal on Computational Optimization*, 10:100024, 2022. ISSN 2192-4406. **DOI** 10.1016/j.ejco.2021.100024. URL http://dx.doi.org/10.1016/j.ejco.2021.100024.

Pravin Chandra and Manoj K. Gupta. Comprehensive survey on data warehousing research. *International Journal of Information Technology*, 10(2):217–224, December 2017.

Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Rec.*, 26(1):65–74, mar 1997.

Ali Danandeh Mehr, Babak Vaheddoost, and Babak Mohammadi. Enn-sa: A novel neuro-annealing model for multi-station drought prediction. *Computers amp; Geosciences*, 145:104622, December 2020. ISSN 0098-3004. **DOI** 10.1016/j.cageo.2020.104622. URL http://dx.doi.org/10.1016/j.cageo.2020.104622.

Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. Simulated annealing: From basics to applications. In *International Series in Operations Research & Management Science*, pages 1–35. Springer International Publishing, September 2018.

Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. Modeling metadata in data lakes—A generic model. *Data & Knowledge Engineering*, 136:101931, 2021.

Huang Fang. Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 820–824. IEEE, 2015.

Danilo Fernandes, Douglas L. L. Moura, Gean Santos, Geymerson S. Ramos, Fabiane Queiroz, and Andre L. L. Aquino. Towards edge-based data lake architecture for intelligent transportation system. In *Proceedings of the Int'l ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, amp; Ubiquitous Networks*, MSWiM '23. ACM, October 2023a. **DOI** 10.1145/3616394.3618270. URL http://dx.doi.org/10.1145/3616394.3618270.

Danilo Fernandes, Geymerson S. Ramos, Rian G. S. Pinheiro, and André L. L. Aquino. Data lake organization source code and datasets. https://github.com/danilofernandes45/Data-Lake-Organization/, jun 2023b.

Raul Fernandez, Essam Mansour, Abdulhakim A. Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *IEEE 34th International Conference on Data Engineering (ICDE)*, pages 989–1000, 2018a.

Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: A data discovery system. In *IEEE 34th International Conference on Data Engineering (ICDE)*, 2018b.

Matteo Francia, Enrico Gallinucci, Matteo Golfarelli, Anna Giulia Leoni, Stefano Rizzi, and Nicola Santolini. Making data platforms smarter with MOSES. *Future Generation Computer Systems*, 125:299–313, December 2021.

Giuseppe Futia, Antonio Vetrò, and Juan Carlos De Martin. SeMi: A SEmantic Modeling machIne to build Knowledge Graphs with graph neural networks. *SoftwareX*, 12:100516, July 2020.

Jerry Gao, Chunli Xie, and Chuanqi Tao. Big data validation and quality assurance–issuses, challenges, and needs. In *2016 IEEE symposium on service-oriented system engineering (SOSE)*, pages 433–441. IEEE, 2016.

Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. Leveraging the Data Lake: Current State and Challenges. In Carlos Ordonez, Il-Yeol Song, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil, editors, *Big Data Analytics and Knowledge Discovery*, pages 179–188. Springer International Publishing, 2019.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN 0201157675.

Alex Gorelik. *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*. O'Reilly Media, 2019.

Amit Gupta, Rémi Lebret, Hamza Harkous, and Karl Aberer. Taxonomy induction using hypernym subsequences. In *ACM Conference on Information and Knowledge Management*, page 1329–1338. Association for Computing Machinery, 2017.

Ibrahim Abaker Targio Hashem, Victor Chang, Nor Badrul Anuar, Kayode Adewole, Ibrar Yaqoob, Abdullah Gani, Ejaz Ahmed, and Haruna Chiroma. The role of big data in smart city. *International Journal of Information Management*, 36(5):748–758, 2016.

Wu He and Li Da Xu. Integration of Distributed Enterprise Applications: A Survey. *IEEE Transactions on Industrial Informatics*, 10(1):35–42, 2012.

Jeff Heaton. An Empirical Analysis of Feature Engineering for Predictive Modeling. In *IEEE Region 3 South East Conference (SoutheastCon'16)*, 2016.

IRENA Group. Innovation Landscape Brief: Internet of Things. Technical Report ISBN 978-92-9260-142-3, International Renewable Energy Agency, 2019.

A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, November 1962.

Daniel Kahneman, Olivier Sibony, and Cass R Sunstein. *Noise: A flaw in human judgment*. Little, Brown, 2021.

Yaghoob Karimi, Mostafa Haghi Kashani, Mohammad Akbari, and Ebrahim Mahdipour. Leveraging big data in smart cities: A systematic review. *Concurrency and Computation: Practice and Experience*, 33(21):e6379, 2021.

Evgeny Kharlamov, Luca Giacomelli, Evgeny Sherkhonov, Bernardo Cuenca Grau, Egor V. Kostylev, and Ian Horrocks. Semfacet: Making hard faceted search easier. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, page 2475–2478. Association for Computing Machinery, 2017.

Paul Le Noac'H, Alexandru Costan, and Luc Bougé. A performance evaluation of Apache Kafka in support of big data streaming applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4803–4806. IEEE, 2017.

Wen-Syan Li and Chris Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proceedings of the 20th international conference on very large data bases*, 1994.

Wen-Syan Li, Chris Clifton, and Shu-Yao Liu. Database integration using neural networks: implementation and experiences. *Knowledge and information systems*, 2(1):73–96, 2000.

Ying Li, AiMin Zhang, Xinman Zhang, and Zhihui Wu. A Data Lake Architecture for Monitoring and Diagnosis System of Power Grid. In *Artificial Intelligence and Cloud Computing Conference (AICC'18)*, 2018.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Gustavo V. Machado, Ítalo Cunha, Adriano C. M. Pereira, and Leonardo B. Oliveira. DOD-ETL: distributed on-demand ETL for near real-time business intelligence. *Journal of Internet Services and Applications*, 10(1), November 2019.

Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Soren Auer, and Jens Lehmann. Uniform Access to Multiform Data Lakes Using Semantic Technologies. In *21st International Conference on Information Integration and Web-based Applications & Services (IIWAS'19)*, 2019.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, March 1947.

Vivien Marx. The Big Challenges of Big Data. *Nature*, 498(7453):255–260, 2013.

Christian Mathis. Data Lakes. *Datenbank-Spektrum*, 17(3):289–293, October 2017.

Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, December 2011.

Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table union search on open data. *Proceedings of the VLDB Endowment*, 11(7):813–825, mar 2018.

Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. Organizing Data Lakes for Navigation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, page 1939–1950. Association for Computing Machinery, 2020.

Fatemeh Nargesian, Ken Pu, Bahar Ghadiri-Bashardoost, Erkang Zhu, and Renée J. Miller. Data lake organization. *IEEE Transactions on Knowledge and Data Engineering*, 35(1): 237–250, 2023. **DOI** 10.1109/TKDE.2021.3091101.

Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, 2017.

E.N. Osegi and E.F. Jumbo. Comparative analysis of credit card fraud detection in simulated annealing trained artificial neural network and hierarchical temporal memory. *Machine Learning with Applications*, 6:100080, December 2021. ISSN 2666-8270. **DOI** 10.1016/j.mlwa.2021.100080. URL http://dx.doi.org/10.1016/j.mlwa.2021.100080.

Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Ronin: Data lake exploration. *Proceedings of the VLDB Endowment*, 14(12):2863–2866, oct 2021.

Foster Provost and Tom Fawcett. Data Science and its Relationship to Big Data and Data-driven Decision Making. *Big data*, 1(1):51–59, 2013.

Geymerson S. Ramos, Danilo Fernandes, Jorge Artur P. de M. Coelho, and Andre L. L. Aquino. *Toward Data Lake Technologies for Intelligent Societies and Cities*, pages 3–29. Springer International Publishing, Cham, 2023. ISBN 978-3-031-30514-6.

M. Mazhar Rathore, Awais Ahmad, Anand Paul, and Seungmin Rho. Urban planning and building smart cities based on the Internet of Things using Big Data analytics. *Computer Networks*, 101:63–80, 2016. Industrial Technologies and Applications for the Internet of Things.

David Reinsel, John Gantz, and John Rydning. Data Age 2025: The Digitization of the World From Edge to Core. Technical report, International Data Corporation (IDC), 2018.

Celso C. Ribeiro, Isabel Rosseti, and Reinaldo Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54(2):405–429, August 2011.

Yuya Sasaki. A Survey on IoT Big Data Analytic Systems: Current and Future. *IEEE Internet of Things Journal*, 9(2):1024–1036, 2022.

Pegdwende Sawadogo and Jerome Darmont. On Data Lake Architectures and Metadata Management. *Journal of Intelligent Information Systems*, 56(1):97–120, 2021.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*, 2010.

Socrata. Socrata open data api. https://dev.socrata.com/, sep 2022.

Gürkan Solmaz, Everton Luís Berz, Marzieh Farahani Dolatabadi, Samet Aytaç, Jonathan Fürst, Bin Cheng, and Jos den Ouden. Learn from IoT: Pedestrian Detection and Intention Prediction for Autonomous Driving. In *Proceedings of the 1st ACM Workshop on Emerging Smart Technologies and Infrastructures for Smart Mobility and Sustainability*, page 27–32. Association for Computing Machinery, 2019.

Jerzy Stefanowski, Krzysztof Krawiec, and Robert Wrembel. Exploring Complex and Big Data. *International Journal of Applied Mathematics and Computer Science*, 27(4):669–679, 2017.

Éric D. Taillard, Philippe Waelti, and Jacques Zuber. Few statistical tests for proportions comparison. *European Journal of Operational Research*, 185(3):1336–1350, March 2008.

Marieh Talebkhah, Aduwati Sali, Mohsen Marjani, Meisam Gordan, Shaiful Jahari Hashim, and Fakhrul Zaman Rokhani. IoT and Big Data Applications in Smart Cities: Recent Advances, Challenges, and Critical Issues. *IEEE Access*, 9:55465–55484, 2021.

James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. From natural language processing to neural databases. In *Proceedings of the VLDB Endowment*, pages 1033–1039. VLDB Endowment, 2021.

François Torregrossa, Robin Allesiardo, Vincent Claveau, and Guillaume Gravier. Unsupervised tree extraction in embedding spaces for taxonomy induction. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, page 302–309. Association for Computing Machinery, 2021.

Deepak Vohra. Using apache sqoop. In *Pro Docker*, pages 151–183. Springer, 2016.

Kun Wang, Yanpeng Cui, Jianwei Hu, Yu Zhang, Wei Zhao, and Luming Feng. Cyberbullying detection, based on the fasttext and word similarity schemes. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 20(1), nov 2020.

Han Yu, Hongming Cai, Zhiyuan Liu, Boyi Xu, and Lihong Jiang. An Automated Metadata Generation Method for Data Lake of Industrial WoT Applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–14, 2021a.

Shangdi Yu, Yiqiu Wang, Yan Gu, Laxman Dhulipala, and Julian Shun. Parchain: A framework for parallel hierarchical agglomerative clustering using nearest-neighbor chain. *Proceedings of the VLDB Endowment*, 15(2):285–298, oct 2021b.

Vincent F. Yu and Shih-Wei Lin. Multi-start simulated annealing heuristic for the location routing problem with simultaneous pickup and delivery. *Applied Soft Computing*, 24:284–290, 2014.

Elisabeta Zagan and Mirela Danubianu. Data Lake Approaches: A Survey. In *International Conference on Development and Application Systems (DAS'20)*, 2020.

Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. Lsh ensemble: Internet-scale domain search. *Proceedings of the VLDB Endowment*, 9(12):1185–1196, aug 2016.