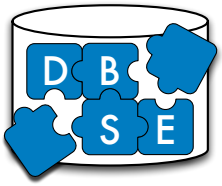
Otto-von-Guericke-Universität Magdeburg
Faculty of Computer Science



Databases
and
Software
Engineering

Master's Thesis

Evaluation of Unsupervised and Reinforcement Learning approaches for Horizontal Fragmentation

Authors:

Milena Malysheva, Ivan Prymak

October 29th, 2019

Advisors:

Prof. Dr. rer. nat. habil. Gunter Saake
M.Sc. Gabriel Campero

Databases and Software Engineering Workgroup,
University of Magdeburg

Malysheva, Milena; Prymak, Ivan:
Evaluation of Unsupervised and Reinforcement Learning approaches for Horizontal Fragmentation

Master's Thesis, Otto-von-Guericke-Universität Magdeburg
Faculty of Computer Science, 2019.

Abstract

Finding the right horizontal fragmentation is one of the core database tuning tasks, helping to improve query performance by reducing the data accessed or improving parallelism. More so than any other physical database design tasks, choosing a "good" or optimal horizontal fragmentation scheme has a high computational complexity. One reason for that lies in the fact that, unlike the similar physical design tasks of vertical fragmentation or indexing, where the search space is limited by a relatively small number of table attributes, the complexity of horizontal fragmentation depends on the number of tuples in a table. Most common algorithms solving the task of horizontal fragmentation use a set of empirical heuristic rules [1, 2], which help them prune the search space, but cannot guarantee optimality for all cases.

This thesis proposes two alternative Machine-Learning-based horizontal fragmentation solutions (clustering-based and Reinforcement-Learning-based), which seek to extend the potential of existing solutions. The first of them is developed to be a simple and transparent intermediate ML-based baseline for evaluating the second. The RL-based approach is built to be continuously trained on the new workloads and to generate fragmentation schemes very fast after the training phase.

In this work we test different configuration parameters for the clustering based solution: cost model usage, similarity measures and linkage criteria. As for the Reinforcement-Learning-based solution, we evaluate multiple Deep-RL agents in the case of fixed query workload and in the case of generalized query workload. We also test the impact of hyper-parameters on the agent convergence. Both approaches are compared with a classical horizontal fragmentation algorithm proposed by Zhang and Orlowska [1]. This thesis provides a multifaceted comparative analysis of the proposed approaches and suggests directions for further improvements.

Acknowledgements

We would like to express our profound gratitude to our thesis supervisor M.Sc. Gabriel Campero Durand. He was a constant source of support and advice and motivated us to look for new approaches to solving the problems. He guided us through the process and was always available when we needed his help.

We would also like to thank Prof. Dr. rer. nat. habil. Gunter Saake for letting us write our Master thesis at his chair. We would like to acknowledge Dr.-Ing. Christoph Steup for giving us valuable advice at our kickoff meeting.

And last but not least, we would like to thank our families and friends back home for their love and encouragement. We are especially thankful to our friend Guzel Mussilova for providing a feedback on the drafts of our thesis.

Declaration of Academic Integrity

We hereby declare that this thesis is solely our own work and we have cited all external sources used.
Magdeburg, October 29th 2019

_____

Milena Malysheva

_____

Ivan Prymak

Contents

List of Figures **4**

List of Tables **5**

1 Introduction (MM, IP) **1**
1.1 Motivation . . . 1
1.2 Research aim . . . 4
1.3 Research methodology . . . 4
1.4 Thesis structure . . . 5

2 Background **7**
2.1 Physical database design (IP) . . . 7
2.1.1 Vertical fragmentation . . . 10
2.1.2 Horizontal fragmentation . . . 10
2.1.3 Hybrid fragmentation . . . 11
2.2 Algorithms for horizontal fragmentation (MM) . . . 11
2.3 Machine learning . . . 22
2.3.1 Supervised learning (IP) . . . 23
2.3.2 Unsupervised learning (MM) . . . 23
2.3.2.1 Hierarchical-based clustering . . . 30
2.3.2.2 Clustering-based horizontal fragmentation . . . 35
2.3.3 Reinforcement learning (IP) . . . 43
2.3.3.1 Basic RL . . . 44
2.3.3.2 Deep RL . . . 49
2.4 Summary . . . 53

3 Prototype implementation and research questions **55**
3.1 Research questions . . . 55
3.2 General structure (MM) . . . 56
3.3 Cost model selection (IP) . . . 56
3.4 Classical algorithm adaptation (MM) . . . 59
3.5 Clustering-based solution (MM) . . . 60
3.5.1 Specific research questions . . . 60
3.5.2 Clustering algorithm selection . . . 60

3.5.3 Input data representation . . . 63
3.5.4 Similarity measures and linkage criteria . . . 68
3.6 Deep-RL-based solution (IP) . . . 69
3.6.1 Research questions . . . 69
3.6.2 Architecture . . . 70
3.6.3 Input data representation . . . 71
3.6.4 Action representation . . . 74
3.6.5 Action pruning . . . 76
3.7 Summary . . . 77

4 Experimental design (MM, IP) 79
4.1 Experimental environment . . . 79
4.2 Dataset . . . 80
4.3 Workloads . . . 81
4.4 Algorithms settings . . . 82
4.5 Summary . . . 84

5 Evaluation and Results 85
5.1 Research questions . . . 85
5.2 Clustering-based solution (MM) . . . 86
5.2.1 No cost model included . . . 88
5.2.2 With the cost model included . . . 89
5.3 Deep-RL-based solution (IP) . . . 92
5.3.1 Convergence in the case of fixed workload . . . 93
5.3.2 Convergence in the case of generalized workload . . . 96
5.4 Comparison of the solutions . . . 99
5.4.1 Quality of the results (MM, IP) . . . 99
5.4.2 Number of cost model calls (MM) . . . 101
5.4.3 Inference times (IP) . . . 103
5.5 Summary . . . 103

6 Related work and Future Directions 105
6.1 Clustering for physical design problems (MM) . . . 105
6.2 RL for physical design problems (IP) . . . 107
6.3 Other optimization algorithms for physical design problems (MM) . . . 109
6.4 Summary . . . 112

7 Conclusion and Future work (MM, IP) 113
7.1 Work summary . . . 113
7.2 Threats to validity . . . 115
7.3 Future work . . . 116

Bibliography 119

List of Figures

1.1 CRISP-DM process diagram 5

2.1 A comparison between logical and physical design 8
2.2 Taxonomy of fragmentation strategies 9
2.3 Example of vertical fragmentation 10
2.4 Example of horizontal fragmentation 11
2.5 Example of hybrid fragmentation 12
2.6 Taxonomy of horizontal fragmentation algorithms 13
2.7 A taxonomy of ML approaches 23
2.8 Example of clustered data 24
2.9 Taxonomy of clustering algorithms 26
2.10 DBSCAN clustering mechanism 28
2.11 Results of clustering with Gaussian mixture modeling (a) and k-means (b) algorithms (k = 3) 30
2.12 Results of clustering with commonly used clustering algorithms 30
2.13 Agglomerative and divisive hierarchical clustering 31
2.14 Dendrogram and clusters generated for k = 5 32
2.15 Single-linkage criterion 32
2.16 Complete-linkage criterion 33
2.17 Average-linkage criterion 33
2.18 Centroid-linkage criterion 33
2.19 Agent-environment interaction in RL 43
2.20 Taxonomy of modern RL algorithms 50

2.21 Network architectures for DQN and recent distributional RL algorithms . 53

3.1 General structure of the software system 56
3.2 Input data in Euclidean space 66
3.3 Architecture of Deep-RL based solution 70
3.4 Schematic representation of rewards throughout the episode 76
3.5 Schematic representation of action pruning embedded into Deep-RL agent 76

5.1 Profiling results (the clustering-based solution) 87
5.2 Profiling results (the classical fragmentation approach) 88
5.3 Clustering-based solution without cost model usage (execution costs are measured in tuples fetched) 88
5.4 Clustering-based solution with the cost model usage (execution costs are measured in tuples fetched) 90
5.5 Clustering-based solution with the cost model usage (number of cost model calls) 90
5.6 Pareto-optimal solutions (execution costs are measured in tuples fetched) 92
5.7 Rainbow DQN in the case of fixed workload (execution costs are measured in tuples fetched) 94
5.8 Implicit Quantile in the case of fixed workload (execution costs are measured in tuples fetched) 95
5.9 Rainbow DQN in the case of generalized workload (execution costs are measured in tuples fetched) 97
5.10 Implicit Quantile in the case of generalized workload (execution costs are measured in tuples fetched) 98
5.11 Quality of the fragmentation schemes generated by the horizontal fragmentation solutions (execution costs are measured in tuples fetched) . . . 100
5.12 Number of the cost model calls of the clustering-based and the classical fragmentation solutions 102

6.1 Horizontal fragmentation approach proposed by Amina et al. 107
6.2 Overview of DRL-based approach to Learn a Partitioning Advisor 108
6.3 ReJOIN framework 109

List of Tables

2.1 Stirling numbers of the second kind 12
2.2 Predicate usage matrix 18
2.3 Predicate affinity matrix 19
2.4 Ordered predicate affinity matrix 19
2.5 Clustered predicate affinity matrix 20
2.6 Clustering in horizontal fragmentation 1 38
2.7 Clustering in horizontal fragmentation 2 42

3.1 Predicate usage matrix 64
3.2 Atomic fragments 65
3.3 Atomic fragments after processing 66
3.4 Penalty-based method 69
3.5 Example of observation space for 2 queries and 4 maximum fragments . . 74
3.6 Example of action based on query predicate 75

4.1 Ranges of values used by the query generator for numeric fields 82
4.2 Ranges of values used by the query generator for date fields 82
4.3 Parameters of Dopamine agents of DQN family 84

5.5 Average number of initial atomic fragments 102
5.6 Inference times for Rainbow and Implicit Quantile 103
5.7 Average cost model use times 103

7.1 Comparative analysis of the solutions 115

1. Introduction

In this chapter we present the motivation for our work, define research aims, describe the research methodology we adopted and outline the structure of this thesis.

1.1 Motivation

The amounts of data modern applications need to store and process are increasing rapidly. To keep up with this trend, companies and common users require scalable relational database solutions, which make information easy to analyze and accessible via traditional SQL-querying. To allow for fast and efficient database systems, developers of database management systems and database administrators employ various optimization techniques aiming to improve the speed of serving queries.

One core component in tuning a database system for performance is its physical design, which defines the actual structure of the database on disk or in memory. Most of the physical design tasks (index selection, data fragmentation, materialized views, database storage topology) have a goal of increasing throughput and speed of query processing by reducing I/O and memory consumption during runtime.

While solving the tasks of physical database design, it is essential to take into consideration the profile of queries that will be executed on the system. A number of papers agree that it is not feasible to measure performance by a completely random set of queries and the usage of explicit or parametric workflows is wildly recognized when solving physical database design problems [3]. In practice, adjusting the database structure to constantly changing workload parameters proves tricky, leading to degrading performance.

There exist two approaches to automate database configuration for a given query workload. The first one relies on using partial automation, i.e. some database tuning advisor software that suggests a configuration given the database structure and query workload. This approach is prone to inaccurate estimates and exhibiting highly unpredictable

behavior in certain cases [4]. The second approach is the use of fully automated solutions. These solutions can be based on heuristics, that means that they rely on domain specific knowledge and do not employ machine learning methods; or they can more sophisticated machine learning models allowing the configuration management system to learn from its mistakes and successes.

One of the benefits of relational databases is physical data independence. This allows the details of physical layer change without changing the results of query execution. Such changes, however, impact performance. Thus, there exists an opportunity for automated configuration managers to run continuously in a production environment gathering information about query execution and adjusting physical design configuration online.

In this thesis we research on applying machine learning (ML) methods to automate the task of horizontal database fragmentation. This means, finding the groups of predicates defining data fragments, which allow for the best query execution performance, while keeping fragments disjoint and the tables reconstructable.

The task of primary horizontal fragmentation presents a specific challenge because of its vast search space. Whereas in vertical fragmentation, solutions are typically bound by the number of columns, the solutions to horizontal fragmentation tasks are virtually unbounded. Therefore, naive approaches like full search have been proved to be infeasible.

There are different types of the classical horizontal fragmentation algorithms (minterm-based [2], affinity-based [1], cost-based [5]), but all of them are based on the fixed set of empirical heuristic rules to limit the search space, and they can hardly be adapted to the real-world database systems. That is why recently ML-based techniques are becoming popular in this area, since ML approaches are widely used for tasks where it is impossible or infeasible to apply fine-tailored traditional algorithms. The nature of ML algorithms also makes them good candidates, as they can be reused across a wide variety of applications and because they are considered to be more flexible and adjustable than complex tailored heuristics.

One possible solution is to adopt machine learning models that would learn from experience and enable an informed exploration of the search space. However, given the vastness of potential machine learning models that can be adapted to the task (with different levels of complexity), it is necessary to define a general data-adaptive (machine learning) baseline, with limited complexity that would enable comparability and ease of implementation.

In this thesis we pursue the goal of studying the applicability of machine learning approaches to the horizontal fragmentation problem and we compare their results with state-of-the-art approaches. In our research we employ two branches of ML for horizontal fragmentation: Unsupervised Learning (UL) and Reinforcement Learning (RL). We omit applying Supervised Learning because getting enough labeled examples to train a model would require using some other algorithm for horizontal fragmentation. That would also

skew the predictions of the model towards mimicking the results of the chosen algorithm, which might prove sub-optimal.

Our RL-based approach uses model which is designed to learn from interactions with the an environment, learning long-term value of actions it is taking. We believe that using this approach makes it easier to navigate the vast search space, such that after model is fully trained, it can make predictions about fragmentation configurations without exploring the complete search space.

However, classic RL-approaches present challenges by themselves. One of the problems arises from the vast search space the model needs to explore: to store the raw log of traversal through such space would require enormous amounts of memory. The other problem is caused by the need to represent the horizontal fragmentation problem in a form that is accessible to the RL-model, i.e. split recommendation of complete fragmentation into step-wise actions in discrete action space.

To solve some of these challenges we select deep reinforcement learning (DRL) as a specific sub-type of RL-approaches. Its main advantage is using deep neural networks (DNN) to gather information about the search space and drastically reduce the amount of memory needed. Using function approximation based on DNN also provides the benefit of estimating the rewards for unvisited states.

In order to carry out our research we make several assumptions:

- We consider the problem of horizontal fragmentation where the maximum fragments is limited by some constant. This limitation is dictated by the requirement of the RL-based approach to have a discrete finite observation space.
- We limit our design to use cost models instead of actual database query execution statistics. The removal of this assumption will definitely require changes to our models' training and evaluation process and might cause longer convergence times due to external factors affecting the database performance. Specifically, we use an HDD cost model that estimates the cost of query execution as the amount of data the database management system will have to fetch from the disk.
- We do not consider the costs of creating and maintaining fragments, focusing solely on improvement of the query execution time.
- We assume that the database on which our solutions work is not distributed. Therefore, we do not consider the costs and characteristics, which are typically associated with distributed databases (i.e. transportation costs, fragment allocation, distributed query planning, etc.)
- We assume that all queries of the workload are executed equally often to simplify the problem definition. To remove this assumption small changes will need to be introduced into the cost calculation. In essence, this means that the cost of executing each query should be multiplied by its weight factor, which represents the frequency of this query being executed.

- For our RL-based approach we assume that all queries are only using range predicates to streamline the encoding of queries. For that reason we use synthetic queries with predicates over columns, which can be range-queried.

The assumptions above define our research's scope and outline the possible directions for future work.

1.2 Research aim

The main goal of this thesis is to study applicability of ML approaches to the problem of finding optimal horizontal fragmentation for a fixed maximum amount of fragments. In our thesis, we would like to address the following questions:

1. To what extent can an efficient, case-specific, transparent (i.e., with limited configuration parameters and reasonably easy to interpret) approach based on Unsupervised learning (UL) lead to cost improvements over a primitive strategy?

2. Which configuration parameters of UL-based approach have the most influence on the result?

3. What is the training cost and impact of parameters for an RL-based approach? How does the choice of Deep-RL model affect the convergence of RL-solution.

4. How well do proposed ML-based approaches perform compared to each other and to state-of-the-art horizontal fragmentation algorithms? What are the optimal parameters for a representative evaluation that would capture the best performance of both RL- and UL-based approaches?

1.3 Research methodology

We adopt the CRISP-DM process model for research and design in our thesis [6]. It is a popular standard for organizing data mining projects. Although our problem is not strictly a data mining task, CRISP-DM is widely used in this related field and its generality makes it appropriate to use as a framework for organizing work on this thesis.

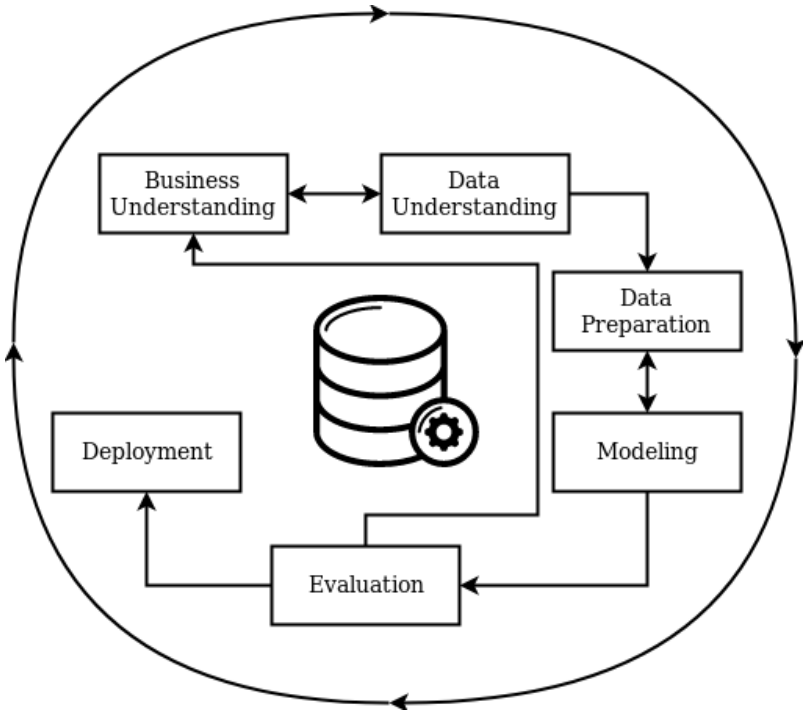


Figure 1.1: CRISP-DM process diagram

Business understanding: At this stage we survey state-of-the-art horizontal fragmentation approaches. We also research existing application of ML to horizontal fragmentation and related fields. This is documented in Chapter 2.

Data understanding: At this stage we study the TPC-H benchmark (tables specifically). Here we decide on what query and table features will ML models use for fragmentation generation. This is documented in Chapter 4.

Data preparation: At this stage we design training and experiment test-benches. This is documented in Chapter 4.

Modeling: At this stage the proposed horizontal fragmentation solutions are designed. This is documented in Chapter 3.

Evaluation: At this stage we compare the performance of the aforementioned ML approaches against each other as well as against traditional horizontal fragmentation algorithms. This is documented in Chapter 5.

Deployment is beyond the scope of this thesis; therefore, we skip this stage.

1.4 Thesis structure

The remainder of this thesis is structured as follows:

- Chapter 2 provides an overview on topics covered in this thesis, such as horizontal database fragmentation and machine learning approaches. We specifically focus on the UL and RL branches of machine learning here.
- Chapter 3 documents the design of the prototypes of the horizontal partitioning advisors developed for this thesis.
- Chapter 4 describes how the experiments are designed. In this chapter we include the description of queries and data used for training and evaluation. We specify the parameters of the UL and RL models used in experiments.
- Chapter 5 includes the experiment results for both UL and RL models against state-of-the-art algorithms as well as against each other. We also provide the interpretation of the results here.
- Chapter 6 consists of our selection of related work that provides context to our study.
- Chapter 7 concludes our work, summarizing the results of our study, providing discussion about possible future work and disclosing possible threats to the validity of our work.

2. Background

In this chapter we provide an overview of the topics covered in this thesis. This chapter is structured as follows:

- In Section 2.1 we present concepts of database physical design, focusing on database fragmentation.
- In Section 2.2 we proceed with describing state-of-the-art algorithms for solving the problem of finding optimal horizontal fragmentations.
- In Section 2.3 we describe machine learning approaches, outlining the three main branches: supervised learning, unsupervised learning and reinforcement learning. We provide an in-depth overview of the last two, as they are relevant for our research.
- In Section 2.4 we summarize our overview.

2.1 Physical database design

Physical database design represents the mapping of a logical data model of a database to the physical data structure of the target database management system (DBMS). Physical database design typically consists of two parts:

- **Pre-deployment**, where the logical data model is translated into table definitions and column constraints, relations are broken down to normalize them, and basic indexing is added.
- **Post-deployment**, where the physical design tuning takes place. Here the goal is to improve performance while keeping higher level abstraction models intact.

It should be noted that both pre- and post-deployment stages can happen on a live database. The key difference here is that the former introduces a reaction to a change in higher level data models, while the latter is independent from logical and conceptual design.

Figure 2.1 outlines the main conceptual differences between logical and physical database design.

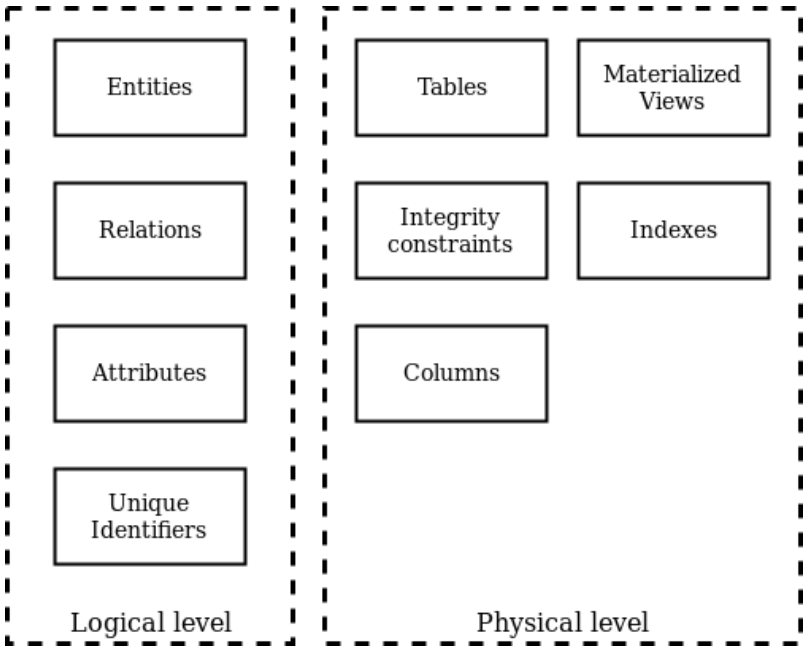


Figure 2.1: A comparison between logical and physical design

- *Tables* are a basic storage structure in physical design of a relational database. They represent typed data stored on a physical storage. Tables consist of multiple tuples (or rows), where each tuple has the same attributes (columns). Tables can be fragmented row-wise or column-wise for manageability or performance reasons.

- *Integrity constraints* serve as an enforcement of application-specific rules on the way data is stored, added, processed and deleted. They also ensure data validity and provide additional metadata for query planners.

- *Materialized views* are cached results of queries or sub-queries, representing a collection of data from one or multiple tables structured in a way that is required by a specific application. These results are stored to reduce the performance impact of repeatedly running costly queries over and over in the cases when an average cost of updating a materialized view is lower than an average cost of re-running the query.

- *Indexes* are optional data structures used in databases to optimize query lookup time for specific columns. There exist multiple types of indexes, tailored to the

data type of the column (or columns) that needs indexing or to the type of the filter predicate (or predicates) in a query that is being optimized.

Among other challenges in database physical performance optimization is the problem of fragmenting the relation into physical fragments, so that their layout produces the best performance improvement over a specified query workload. Better query run time for fragmented relations is achieved by reducing I/O costs, allowing to fetch from disk and store in memory less data. Similarly, fragmentation can also contribute to parallelism.

There are several fragmentation techniques, their taxonomy is presented in Figure 2.2, adapted from [7].

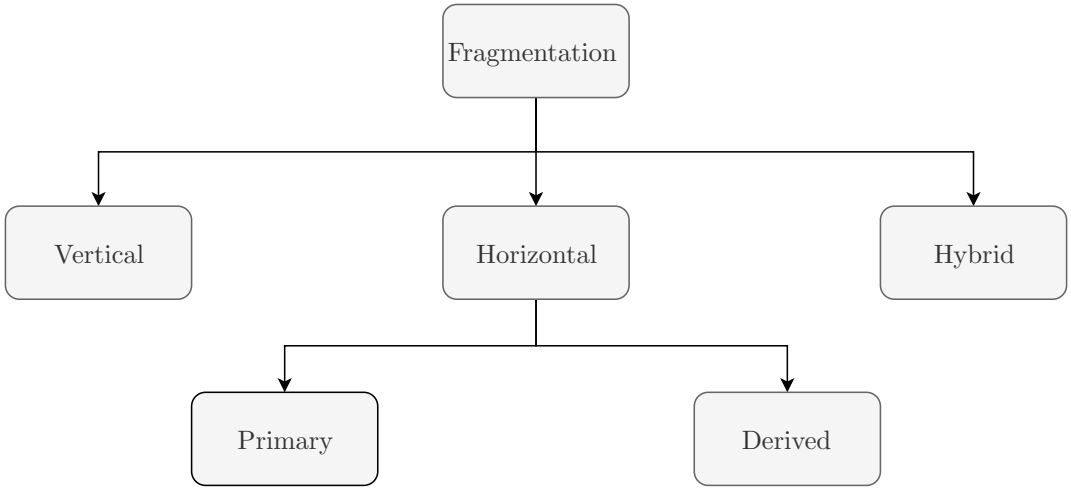


Figure 2.2: Taxonomy of fragmentation strategies

In general, each fragmentation technique is a way of altering the underlying physical data structure while keeping the base logical-level relation intact. To ensure that no data is lost and no redundant data is created during the fragmentation process, each type of fragmentation must comply with the following *correctness rules*:

- **Completeness:** there must be no data loss due to the fragmentation process. Each tuple from the base table should be present in at least one resulting fragment.
- **Reconstructability:** it should be possible to fully reconstruct the tuples of the base table from the information stored in fragments. This ensures preservation of functional dependencies.
- **Disjointness:** there should not be no tuples from the base table which are present in two or more fragments. This is a requirement when there is no replication.

2.1.1 Vertical fragmentation

Vertical fragmentation is a "column-wise" decomposition of a table into multiple sub-tables. This results in fragments consisting of various attribute groups of the original relation. For table R, fragments $R_1, R_2, ..., R_n$ can be formed by applying the following operation:

$$R_i = \Pi_{A_i}(R) + \Pi_{primarykey}(R), (1 \leq i \leq n) \tag{2.1}$$

where A_i is a set of non-key attributes of table R. To satisfy the *completeness* requirement each of the attributes from the original table should be present in at least one set from A. An illustration of fragmenting a table vertically is presented in Figure 2.3.

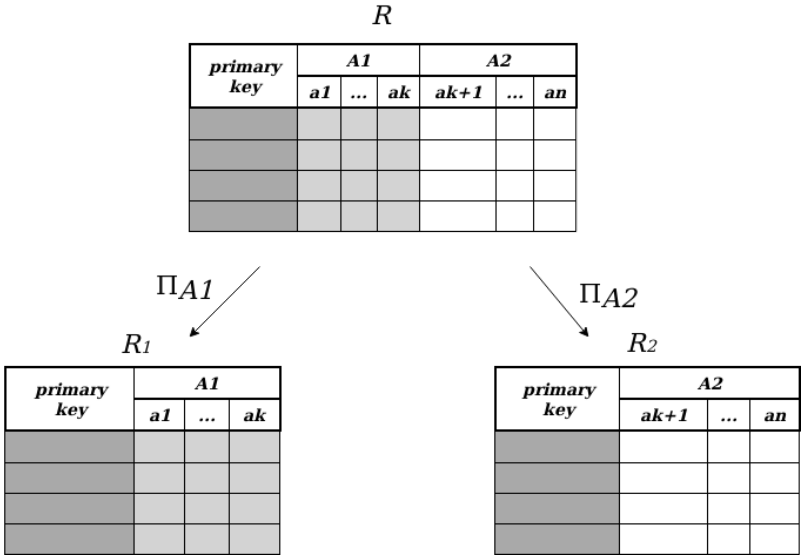


Figure 2.3: Example of vertical fragmentation

The primary key is included in all fragments of a vertically fragmented table to ensure that it is possible to reconstruct the relation satisfying the *reconstructability* requirement. One can reconstruct the original table by joining the fragments on the primary key:

$$R = R_1 \bowtie_{primarykey} R_2 \bowtie_{primarykey} ... \bowtie_{primarykey} R_n \tag{2.2}$$

2.1.2 Horizontal fragmentation

Horizontal fragmentation is a "tuple-wise" decomposition of a table R into sub-tables $R_1, R_2, ..., R_n$, where each sub-table R_i can be expressed as follows [8]:

$$R_i = \sigma_{P_i}(R), (1 \leq i \leq n) \tag{2.3}$$

where P_i are fragmentation predicates. In principle, these predicates should not allow overlapping fragments, and the collection of predicates should be sufficient to reconstruct the whole table.

An illustration of fragmenting a table horizontally can be seen in Figure 2.4.

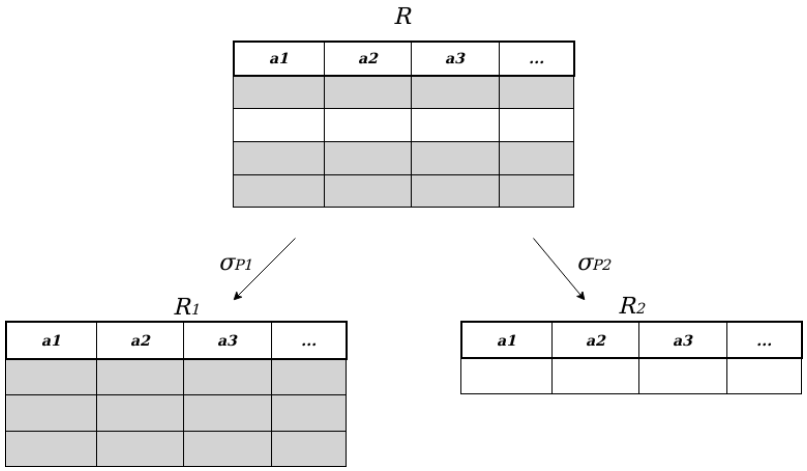


Figure 2.4: Example of horizontal fragmentation

There are two main types of horizontal fragmentation: primary and derived.

Primary horizontal fragmentation partitions the table using its own non-key data to guide the fragmentation process. This improves the performance by allowing the query planner to take advantage of fragment metadata and entirely skip fetching fragments which do not fit the query.

Derived horizontal fragmentation partitions a table using its key data, usually by association to some other table, related to the target table. This approach is useful when the target query is only used together with the related table and the query workload does not have predicates on attributes from the target table.

2.1.3 Hybrid fragmentation

Hybrid (also called mixed) fragmentation is a type of fragmentation, which combines horizontal and vertical fragmentation. The fragments in such case are represented by groups of rows and columns of the original relation.

An example of hybrid fragmentation can be seen in Figure 2.5:

2.2 Algorithms for horizontal fragmentation

As discussed earlier, a proper fragmentation plays an important role in the performance of a database system. However, finding an optimal fragmentation is not an easy task to solve. An obvious choice, which can provide an optimal fragmentation scheme by

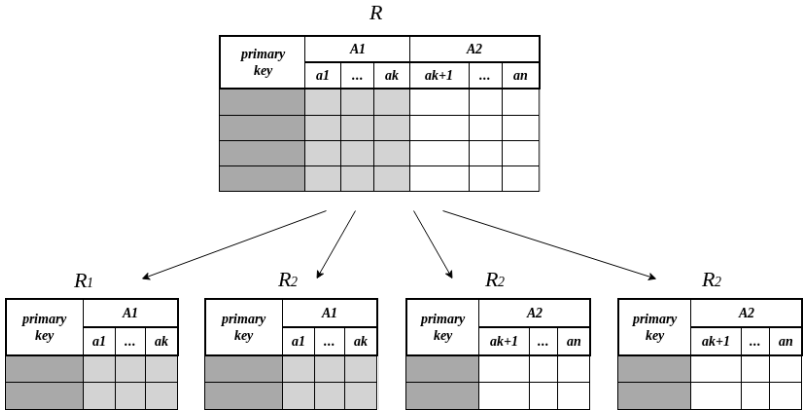


Figure 2.5: Example of hybrid fragmentation

considering all possible fragmentations would be exhaustive search. When possible, it is a good approach to follow, if the task is to perform vertical fragmentation, as the number of attributes in a table is often small and this number limits the search space. However, it is infeasible to solve the horizontal and hybrid fragmentation problems with exhaustive search. In combinatorics, the number of ways to group n elements into k groups is called a Stirling Number of the second kind. It is denoted as $S(n, k)$. It is clear from Table 2.1, that even for small values of n and k, the Stirling number $S(n, k)$ can become very large. For example, for 64 rows and 5 fragments the number of possible fragmentation schemes is equal to $4 * 10^{42}$. Moreover, a table with only 64 rows is an unrealistic scenario. PostgreSQL, for instance, does not put a limit on the number of rows in the table at all.

n \ k	1	2	3	4	5	6	7	8	9	10	11
1	1										
2	1	1									
3	1	3	1								
4	1	7	6	1							
5	1	15	25	10	1						
6	1	31	90	65	15	1					
7	1	63	301	350	140	21	1				
8	1	127	966	1701	1050	266	28	1			
9	1	255	3025	7770	6951	2646	462	36	1		
10	1	511	9330	34105	42525	22827	5880	750	46	1	
11	1	1023	28501	145750	246730	179487	63987	11880	1155	55	1

Table 2.1: Stirling numbers of the second kind

In general, in order to reduce the search space for problems like horizontal fragmentation such methods as branch and bound or dynamic programming are usually applied [9]. But these methods are still not fast enough to provide a "good" solution within a limited time frame. Therefore, nowadays horizontal fragmentation is performed using various heuristic-based algorithms.

A comprehensive taxonomy of the horizontal fragmentation algorithms inspired by Bouchakri et al. [10] is presented in Figure 2.6.

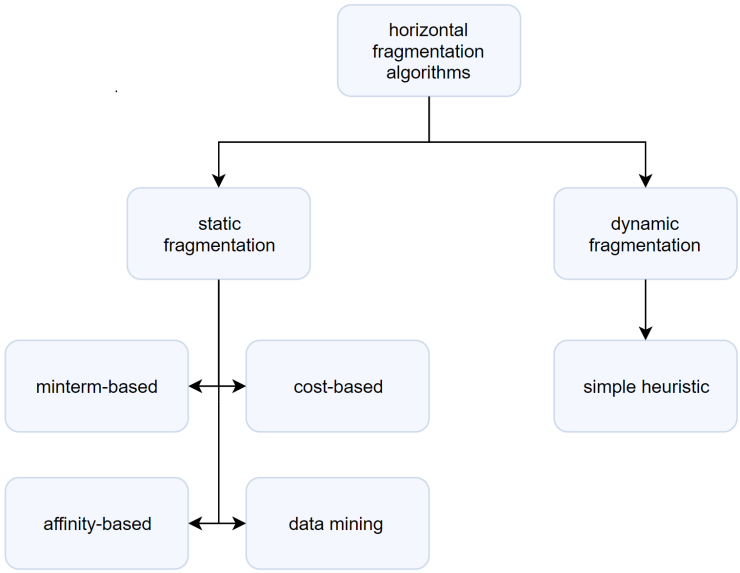


Figure 2.6: Taxonomy of horizontal fragmentation algorithms

The task of horizontal fragmentation can be addressed in two settings: static and dynamic. A static setting corresponds to the cases where the workload (or patterns of the user queries) is known beforehand and does not change over time. However, in real life it is difficult to know user queries at the analysis stage. That is why usually the default setting for the task is dynamic (with changing patterns of user queries). As noted by Ozu et al. [2], the question is not whether a system is dynamic, but rather how dynamic it is. If the system is not very dynamic designers might choose as a workable solution to map a dynamic setting to a corresponding static setting, analysing only the most popular user queries from the database system. This approach will probably not produce an optimal fragmentation scheme, but it will reduce the search space significantly and allow to generate "good enough" solutions. In this context, if the information about all user queries is given and now the queries, which will be an input for a fragmentation algorithm, should be selected, then the "80/20 rule" might be applied as a rule of thumb [2]. This rule reads that the most popular 20% of the queries that tend to make up 80% or more of all queries access should be used as an input for the fragmentation algorithm.

As for the static fragmentation algorithms, all of them usually fall into one of the following categories:

- **Minterm-based algorithms**, which generate a fragmentation scheme from the conjunction of simple predicates [2].
- **Affinity-based algorithms**, which group simple predicates from the queries according to their affinities and generate horizontal fragments using a conjunction of predicates belonging to the same group [1].
- **Data mining algorithms**, which will be discussed later in detail, focusing on the clustering-based solutions.
- **Cost-based algorithms**, which work by generating a set of valid fragmentation schemes and selecting the best one using a cost model. One of such algorithms was proposed by Bellatreche et al. [5].

This taxonomy is neither strict nor entirely exhaustive: e.g. many data mining and affinity-based algorithms use cost models to guide the fragmentation process. In addition, other types of horizontal fragmentation algorithms can be distinguished: graph-based algorithms, algorithms based on integer programming, etc.

Minterm-based algorithms

Ceri, Negri and Pelagatti [8] formulate the horizontal partitioning problem using the access patterns of the applications. Authors define some fundamental terms, which later were used in many other minterm-based fragmentation algorithms. One of these terms is the so-called *simple predicate*. If A_i is an attribute of a relation R, a simple predicate p_k defined on the relation R can be expressed as follows:

$$p_k = A_i \ \theta \ V, \tag{2.4}$$

where V is a value that lies within the domain of the attribute A_i and θ is a comparison operator.

A conjunction of simple predicates in their natural or negated form is called a *minterm predicate*. Minterm predicates are generated from a set of simple predicates, which are selected according to the rules of completeness and minimality.

The *completeness* of a set of simple predicates is formulated using access statistics. A set of simple predicates is said to be complete if tuples that belong to the same fragment (constructed from a minterm predicate) have the same probability of being accessed. Completeness of the predicates set can be checked automatically, but in this case access probabilities of each tuple in relation on each query (application) should be known. That is why a complete set of predicates is often chosen by a designer who uses his own experience and his knowledge about the dataset [2].

A set of predicates is *minimal* if each predicate in the set is *relevant*. A predicate is relevant if there are two fragments f_i and f_j obtained from minterm predicates m_i

and m_j, which differ only in the form of this predicate, and these two fragments have different ratio between number of accesses to their records $acc(m_i)$ and $acc(m_j)$ and the cardinality of the fragments:

$$\frac{acc(f_i)}{card(f_i)} \neq \frac{acc(f_j)}{card(f_j)} \tag{2.5}$$

Based on minterm predicates and the concepts of completeness and minimality, authors propose a methodology for discovering access patterns and formulate the optimal partitioning problem for different application environments. Authors have not proposed a fully-fledged algorithm for horizontal fragmentation, but the concepts formulated in the paper are the base for the complete class of minterm-based horizontal fragmentation algorithms.

For instance, based on these concepts Ozsu and Valduriez [2] have proposed an algorithm called `COM_MIN`, which automatically generates a complete and minimal set of simple predicates Pr' from the initial set Pr of simple predicates. First, the algorithm chooses a relevant predicate and adds it to the set Pr'. Then, it iteratively tries to add other simple predicates, while ensuring the minimality of Pr' at each step. The algorithm is presented below. In the algorithm *Rule 1* ensures that each fragment obtained from Pr is accessed differently by at least one application.

Algorithm 1: COM_MIN algorithm [2]

Input: R: relation; Pr: set of simple predicates
Output: Pr': set of simple predicates
Declare: F: set of minterm fragments
find $p_i \in Pr$ such that p_i partitions R according to *Rule* 1 ;
$Pr' \leftarrow p_i$;
$Pr \leftarrow Pr - p_i$;
$F \leftarrow f_i$;
repeat
  find a $p_j \in Pr$ such that p_j partitions some f_k of Pr' according to *Rule*1;
  $Pr' \leftarrow Pr' \cup p_j$;
  $Pr \leftarrow Pr - p_j$;
  $F \leftarrow F \cup f_j$;
  if $\exists p_k \in Pr'$ *which is not relevant* **then**
    $Pr' \leftarrow Pr' - p_k$;
    $F \leftarrow F - f_k$;
  end if
until Pr' *is complete* ;

For primary horizontal fragmentation `PHORIZONTAL` algorithm was developed (Algorithm 2). The algorithm calls `COM_MIN` algorithm to obtain a complete and minimal set of simple predicates and generates a set of minterm fragments M from it. Then it

tries to reduce the number of minterm fragments by eliminating the redundancy and contradictions from the set M.

Algorithm 2: PHORIZONTAL algorithm [2]

Input: R: relation; Pr: set of simple predicates
Output: M: set of minterm fragments
$Pr' \leftarrow$ `COM_MIN(`R, Pr`)`;
determine the set M of minterm predicates;
determine the set I of implications among $p_i \in Pr'$;
for each $m_i \in M$ **do**
  if m_i *is contradictory according to* I **then**
    $M \leftarrow M - m_i$;
  end if
end for

The `PHORIZONTAL` algorithm, in our opinion, has some restrictions which make it hard to use when dealing with a real database system:

- The implications, that are used to eliminate redundancy and contradictions, are defined according to the semantic properties of the predicates, not according to the database state.
- The proposed implementation does not use any cost model and the algorithm will generate as many fragments as it can, following minimality and completeness rules. So the algorithm does not rate a simple predicate in terms of its efficiency as a fragmentation feature, which makes the algorithm hard to use if there are limitations on the number of fragments.

Affinity-based algorithms

Affinity-based algorithms use the concept of affinity between predicates to guide the horizontal fragmentation process. The classic algorithm of this type was developed by Zhang and Orlowska [1]. In this algorithm the fragmentation process is divided into two phases: primary horizontal fragmentation, using a bond energy algorithm, and derived horizontal fragmentation, which is based on the first phase.

The bond energy algorithm was originally introduced for the data fragmentation problem by McCormick and Schwietzer [11]. This algorithm has been already applied to the vertical fragmentation problem before [12]. In their work Zhang and Orlowska adapted the bond energy algorithm to the horizontal fragmentation problem. To support this kind of adaptation, the authors introduce the concept of predicate affinity, which reflects the similarity between two simple predicates in terms of their usage in transactions. Predicates with a high affinity between each other should be grouped together.

In the first phase of the algorithm predicates are clustered using a predicate affinity matrix. The bond energy algorithm permutes rows and columns of the affinity matrix while trying to maximize an overall global affinity measure. After predicates have been ordered, `split-non-overlap` algorithm [12] iteratively splits the matrix into two parts.

In detail, the algorithm has the following steps:

1. Construct the predicate usage matrix with the transactions as rows and simple predicates as columns:

$$PU_{ij} = \begin{cases} 1, \text{if a transaction } i \text{ uses a simple predicate } j \\ 0, \text{otherwise} \end{cases} \tag{2.6}$$

2. Construct the predicate affinity matrix (PA), where a cell (i, j) contains the combined frequency of the transactions, which access predicate i and predicate j.

3. Order the predicates using the bond energy algorithm. The global affinity measure (AM) that needs to be maximized is:

$$\sum_{i=1}^{i=n} \sum_{j=1}^{j=n} PA_{ij} * (PA_{i,j-1} + PA_{i,j+1}), \tag{2.7}$$

   where n is the number of simple predicates and $PA_{i,0} = PA_{i,n+1} = 0$

4. Partition the predicate affinity matrix into non-overlapping fragments with the `split-non-overlap` algorithm. In this step a point x along the main diagonal of the PA matrix is found. It divides the matrix into two parts: upper and lower. According to Navathe et al. [12] the point can be selected by evaluating the cost function:

$$cost = CL * CU - CI^2, \tag{2.8}$$

   where CL is the sum of accesses of transactions which use only predicates from the upper part of the matrix; CU is the sum of accesses of transactions which use only predicates from the lower part; CI is the sum of accesses of transactions which use predicates both from upper and lower parts.

   By minimizing the proposed cost function authors intend to help the process of partitioning the predicate space into two, while achieving as little transactions as possible that have predicates in both of the obtained partitions.

   It should be noted that the cost can also be obtained using other cost functions.

   One problem of this algorithm is that without modifications it will not be able to find "inner" fragments, if there are any (since it only finds one point, that divides the matrix into two parts). This problem can be solved by using the `SHIFT`

procedure [12], which iteratively shifts columns (and rows accordingly) of the PA matrix one position to the right, and then runs the `split-non-overlap` algorithm on the matrix again. The `SHIFT` procedure is called n times to obtain the best possible position of the point x.

To obtain m fragments $m-1$ points should be selected along the main diagonal. This can be achieved by following one of the approaches:

- Try all possible $m-1$ points along the main diagonal. The complexity of such approach is:

$$\binom{n}{m-1} = \frac{n!}{(m-1)*(n-m+1)} \tag{2.9}$$

- Recursively split each partition obtained in the previous iteration into two parts. The complexity of the approach is much lower, but it can lead to suboptimal solutions.

5. Construct fragments using partitions obtained in the previous step.

6. Add to the list of fragments a fragment which is obtained from the negation of the disjunction of the predicate terms.

For illustration, let's consider the following PU matrix presented in Table 2.2.

predicate / transaction	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	Access freq.
T1		1	1							1		5
T2	1					1		1		1	1	20
T3				1	1		1		1		1	20
T4		1	1		1				1			15
T5	1										1	10
T6				1					1	1	1	10

Table 2.2: Predicate usage matrix

The PA matrix, which will be generated in the second step of the algorithm, is presented in Table 2.3.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
P1	30	0	0	0	0	20	0	20	0	20	30
P2	0	20	20	0	15	0	0	0	15	5	0
P3	0	20	20	0	15	0	0	0	15	5	0
P4	0	0	0	30	20	0	20	0	30	10	30
P5	0	15	15	20	35	0	20	0	35	0	20
P6	20	0	0	0	0	20	0	20	0	20	20
P7	0	0	0	20	20	0	20	0	20	0	20
P8	20	0	0	0	0	20	0	20	0	20	20
P9	0	15	15	30	35	0	20	0	45	10	30
P10	20	5	5	10	0	20	0	20	10	35	30
P11	30	0	0	30	20	20	20	20	30	30	60

Table 2.3: Predicate affinity matrix

The output of the bond energy algorithm for the PA matrix is presented in table Table 2.4.

	P6	P1	P8	P10	P11	P7	P4	P9	P5	P2	P3
P6	20	20	20	20	20	0	0	0	0	0	0
P1	20	30	20	20	30	0	0	0	0	0	0
P8	20	20	20	20	20	0	0	0	0	0	0
P10	20	20	20	35	30	0	10	10	0	5	5
P11	20	30	20	30	60	20	30	30	20	0	0
P7	0	0	0	0	20	20	20	20	20	0	0
P4	0	0	0	10	30	20	30	30	20	0	0
P9	0	0	0	10	30	20	30	45	35	15	15
P5	0	0	0	0	20	20	20	35	35	15	15
P2	0	0	0	5	0	0	0	15	15	20	20
P3	0	0	0	5	0	0	0	15	15	20	20

Table 2.4: Ordered predicate affinity matrix

The actual behavior of the `split-non-overlap` algorithm in splitting the matrix will depend on the cost function and on the way in which m-way partitioning is performed. But visually three overlapping clusters can be distinguished:

	P6	P1	P8	P10	P11	P7	P4	P9	P5	P2	P3
P6	20	20	20	20	20	0	0	0	0	0	0
P1	20	30	20	20	30	0	0	0	0	0	0
P8	20	20	20	20	20	0	0	0	0	0	0
P10	20	20	20	35	30	0	10	10	0	5	5
P11	20	30	20	30	60	20	30	30	20	0	0
P7	0	0	0	0	20	20	20	20	20	0	0
P4	0	0	0	10	30	20	30	30	20	0	0
P9	0	0	0	10	30	20	30	45	35	15	15
P5	0	0	0	0	20	20	20	35	35	15	15
P2	0	0	0	5	0	0	0	15	15	20	20
P3	0	0	0	5	0	0	0	15	15	20	20

Table 2.5: Clustered predicate affinity matrix

The concept of predicate affinity was also used by Navathe, Karlapalem and Ra [13]. But instead of using the bond energy algorithm, authors use a graph-based algorithm inspired by the `MAKE-PARTITIONING` vertical fragmentation algorithm proposed by Navathe and Ra [14] to group the predicates. This algorithm takes the predicate affinity matrix in the form of a complete graph as input and produces a linearly connected spanning tree. It generates fragments which are formed by the cycles in the graph.

Along with the minterm-based `PHORIZONTAL` algorithm, the algorithm proposed by Zhang and Orlowska is considered to be a classical horizontal fragmentation approach. It is often used as a baseline for other horizontal fragmentation algorithms. But all the affinity-based horizontal fragmentation algorithms presented above have some drawbacks:

- Transactions do not necessarily consist of an overlapping set of simple predicates. In reality, it could be that there is not even a single pair of transactions sharing a predicate. Moreover, some of the predicates, which are syntactically different, could access the same set of tuples. The algorithms do not handle such cases properly.
- The obtained fragments are not necessarily disjoint. There could be rows which should be placed in several fragments.

Other horizontal fragmentation algorithms

There are other approaches to horizontal fragmentation presented in the literature.

An interesting approach to horizontal fragmentation was developed by Shin and Irani [15]. This approach is based on user reference clusters. Authors claim that the information about user queries is not enough to produce a good horizontal fragmentation scheme. Authors propose to use knowledge about data itself to revise user queries, so that they can be used for better estimation of user reference clusters. This is done by a so-called knowledge-based system. User reference clusters are thus the base for horizontal fragmentation. In the paper authors answer the questions on how knowledge about data can be incorporated into the horizontal fragmentation process, what knowledge is relevant for the task and how the knowledge should be represented.

A transaction-based horizontal fragmentation algorithm was proposed by Khalil et a. [16]. They use an optimal binary fragmentation algorithm, which is based on the branch and bound algorithm. The algorithm takes as input predicate usage matrix and generates two fragments first. Then they can be further split using the same algorithm until the required number of fragments is reached. Authors also proposed a replication protocol, that provides better reliability and availability.

In 2010 Khan and Hoque [17] presented a fragmentation technique, which does not use empirical data such as user queries and their frequencies. The proposed technique can be used at the initial stage of the fragmentation and allocation. The information used in the algorithm can be gathered at the requirements analysis stage: the fragmentation scheme is generated based on the importance of the data with respect to nodes of a distributed system.

Taft et. al., within the context of their solution E-STORE, frame the horizontal fragmentation problem as an integer programming task [18]. Authors specifically tackle fine-grained partitioning for OLTP workloads. In this context they discuss that fine-grained partitioning is important in periods of high query loads, where the distribution of queries needs optimization for load balancing, whereas coarse-grained partitioning can be good enough for periods of moderate query loads. Authors note that fine-grained partitioning needs more precise tracking of accesses and also requires more complex models, than for the case of coarse-grained partitioning. Consequently they propose to use a default coarse-grained approach and only switch to a fine-grained approach when the load is detected to be increasing beyond a threshold. In the case of such increases, authors carry out tracking at tuple-level, to identify better the relevance of tuples to the workloads. At this stage, the authors establish the fragmentation task as a bin-packing/integer programming problem, which consists of assigning tuples into a number of fragments, for good data distribution and load balancing, subject to the constraint that each fragment can receive a maximum load. Since exact solutions to this problem are time consuming, authors study the use of some heuristics such as first fit or greedy, which are shown to provide acceptable approximate solutions.

Curino et al. frame horizontal fragmentation as a graph partitioning problem [19]. They specifically propose SCHISM, a methodology where tuples (or groups of tuples) are

represented as nodes of a graph, and transactions are represented as a set of edges connecting the tuples that the transaction uses. In this way, a min-cut partitioning is sought, in order to create a horizontal fragmentation applicable for distributed scenarios, where the chance of a distributed transaction is reduced and the load is balanced between the nodes. In addition, authors consider data replication. Two drawbacks of this method are: in the first place, the complexity of the partitioning problem as the number of tuples and transactions grow, and in the second place, the difficulty of creating adequate predicates to describe the resulting partitions. Authors adopt and evaluate solutions for such drawbacks, including methods for grouping tuples together and sampling the transactions, and a technique based on a decision tree to identify a compact set of predicates able to describe the fragments to which tuples are assigned.

Golab et al. study different aspects of the approach proposed of Curino et al., such as considering the impact of replication, materialized views, and others [20]. Authors show the interesting connection that reducing the problem to graph-partitioning is more efficient than treating it as an integer programming problem.

Serafini et al. propose CLAY [21], a graph-based approach similar to SCHISM. Authors propose to start with an initial configuration, and to use the graph model for live re-configurations, improving load balancing. CLAY represents a solution different from SCHISM, because authors consider an incremental re-partitioning problem and there is a goal of minimizing the migrations between partitions.

Horticulture [22], by Pavlo et al., is a related work that uses access graphs as models to summarize database traces and to guide the choice of whether to apply horizontal fragmentation among a set of optimizations from an overall physical design process. The work focuses on skew conditions and tackles physical design beyond horizontal fragmentation. It proposes a search tree to evaluate options, a local search approach that gives an approximate solution and a cost model to guide the process. Unlike other research, this work does not consider fine-grained fragmentation, and the use of the graph is limited to data representation.

Apart from the discussed approaches, there are many others aiming to improve existing algorithms with a faster runtime, better results or simpler-to-maintain fragmentation strategies. In the next chapters we discuss clustering-based solutions and reinforcement-learning based solutions, which are the core topic of our research.

2.3 Machine learning

In this section we briefly introduce the area of machine learning (ML), while describing their applicability to the task of horizontal fragmentation. Given that the topic is vast, we focus on the approaches we use in this thesis and briefly mention the other approaches alongside with the reasoning about why we did not choose them.

Machine learning is often defined as a study of algorithms and approaches that computer systems can use to solve tasks without being explicitly programmed [23]. ML approaches rely heavily on mathematical/statistical models to make predictions and decisions.

ML approaches are widely used for tasks where it is impossible or infeasible to apply fine-tailored traditional algorithms. The nature of ML algorithms also makes them good, "pluggable" solutions, meaning that they can be potentially reused across a wide variety of applications.

A taxonomy of ML approaches is presented in Figure 2.7:

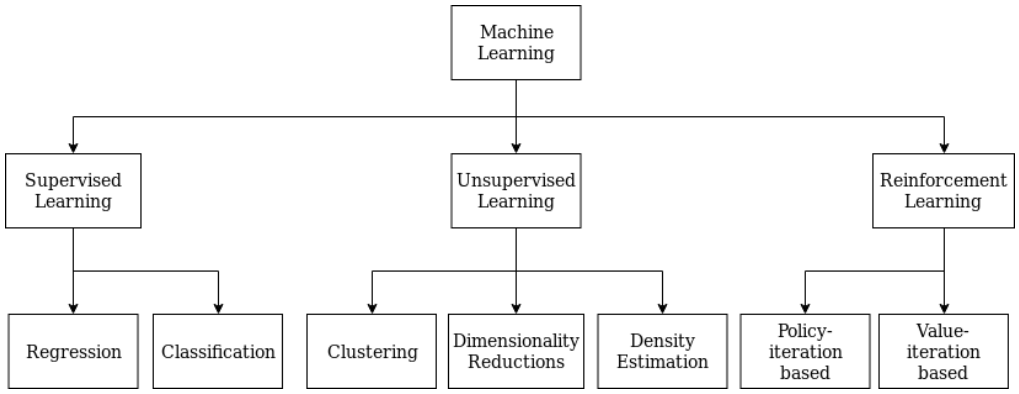


Figure 2.7: A taxonomy of ML approaches

2.3.1 Supervised learning

Supervised learning is a type of machine learning, where the algorithm is designed to approximate a function that maps values from input space to output space, using already mapped examples. These examples are called a *training set*. A supervised learning algorithm produces an inferred mapping function, which can be used to map new, previously unseen by the model, inputs.

The core problem of designing a supervised learning solution for horizontal fragmentation problem is preparing a training set of optimally partitioned tables. As discussed earlier, the horizontal partitioning problem proven to be NP-complete; therefore, it is infeasible to gather training sets via naive full search approaches. Employing various traditional heuristic approaches to generate training data would implicitly change the task from "finding an optimal horizontal fragmentation" to "finding the horizontal fragmentation that traditional approach used in generation of training data would generate", causing the solution to mimic the behaviour and shortcomings of said algorithm.

2.3.2 Unsupervised learning

Unsupervised learning is defined as a type of machine learning, designed to find patterns in data without using a pre-labeled training set. Unlike supervised, unsupervised learning does not have any external guidance and its goal is to identify patterns in a set of inputs to categorize them in some way, which could be very useful for such tasks as horizontal fragmentation.

Density estimation is sometimes referred to as a statistic approach to unsupervised learning [24]. Density estimation task is a task of finding an estimate of underlying

probability density function from a dataset. Density estimation is often used in anomaly detection tasks, distribution of disease cases, for exploratory data analysis, etc. And although it is not suitable for our problem, density estimation can be applied for various clustering-related issues, such as estimation of an optimal number of clusters or outlier detection.

Another important unsupervised learning task is called dimensionality reduction. Due to the curse of dimensionality and high computation complexity of some ML-algorithms it is common practice to reduce a set of features used in the algorithms so that it still captures important patterns in the dataset, but does not contain redundant or irrelevant information. There are some methods of dimensionality reduction aimed at supervised learning, but most of them are unsupervised [24]. In this thesis we do not concentrate on the scalability of our approaches (e.g. scenario of horizontal fragmentation for large workloads) so we will not apply this technique. However, it might be an interesting direction for future work.

Clustering is another well-known unsupervised learning task. The goal of a clustering algorithm is to group given objects into clusters (Figure 2.8). Objects inside each cluster should be more similar to each other than to the objects from other clusters according to a specified metric.

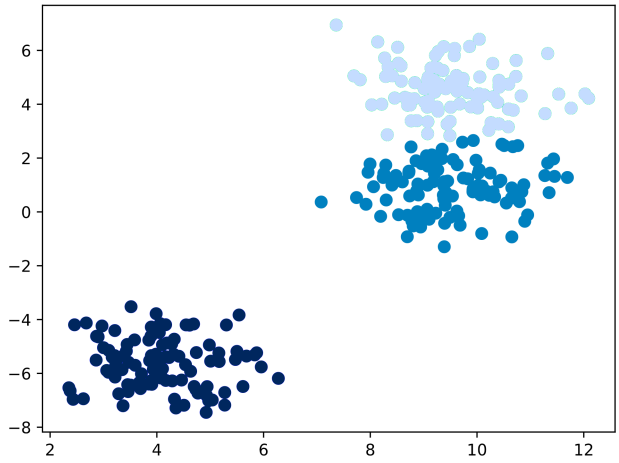


Figure 2.8: Example of clustered data

Applications of clustering analysis include, among others, biology, medicine, archaeology, business analytic, marketing, physical geography. Clustering algorithms are also used in computer science applications, such as: natural language processing, image processing, document clustering and recommender systems.

Clustering-based approaches have been commonly applied for horizontal fragmentation. There are several reasons for it:

- The horizontal fragmentation problem can be easily formulated as a typical clustering problem: objects (rows) need to be grouped so that the objects (rows)

inside one group (fragment) are more similar (in terms of access patterns) to each other than to the objects (rows) in other groups (fragments). The similarity between a pair of rows can be evaluated using query access statistics.

- Clustering is an unsupervised ML problem. So unlike supervised learning there is no need for a training dataset with the correct answers (labels) for given inputs. This is a crucial point since the horizontal fragmentation problem is a computationally complex problem and it is arguably difficult to generate a sufficient amount of input-output pairs for supervised learning.

- Furthermore, unlike reinforcement learning (and its subclass, deep reinforcement learning), unsupervised learning does not need so much time for training. And although a trained model with reinforcement learning could give the correct answer much faster, the amount of time one would spend on training it is potentially very large.

Since we also use a clustering algorithm for horizontal fragmentation in this thesis, we will expand more on this topic.

To solve a clustering task the following steps should be taken:

1. Selection of the dataset.

   A set of objects for clustering should be selected.

2. Selection of the features.

   Finding the proper features for clustering can be pretty challenging in itself. An engineer needs to carefully analyse the points of concern and try to eliminate redundant and irrelevant features. This step may also include normalization procedures.

3. Selection of the similarity measure.

   To assess the similarity between two objects a special function, called similarity measure or similarity function, should be defined.

4. Selection of the clustering criterion.

   Depending on the clustering algorithm some cost model or set of conditions should be specified.

5. Selection of the clustering algorithm.

   A lot of algorithms have been developed for solving clustering tasks. A taxonomy of the clustering algorithms is presented below.

6. Running the algorithm and analysis of the results.

Apart from the aforementioned steps, there are also algorithm-specific questions regarding configurable aspects or hyper-parameters that need to be answered additionally, e.g. how to initialize clusters in k-means algorithm, how to select an appropriate linkage criterion for hierarchical clustering or how to pick a suitable minimum cluster size and radius of a neighborhood for DBSCAN algorithm.

The categorization of clustering algorithms can be performed using different criteria, like the clustering model or relationships between clusters. In this thesis we use a categorization framework, which considers the algorithms from an engineer's point of view and categorizes them based on the general clustering approach used [25]:

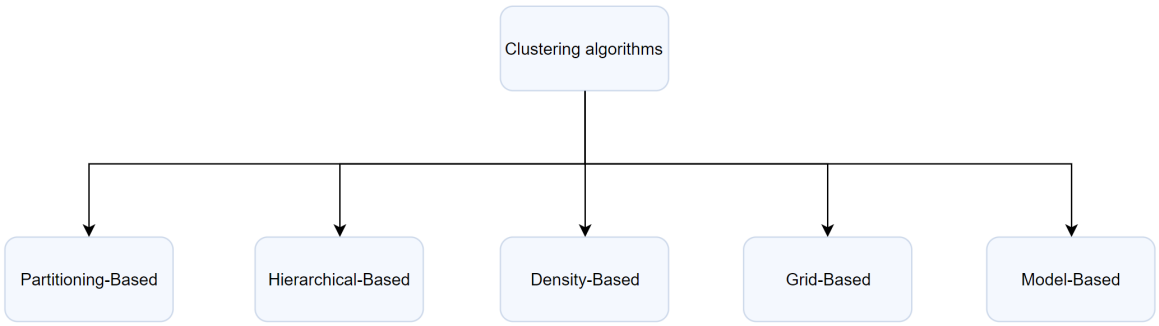


Figure 2.9: Taxonomy of clustering algorithms

Partitioning-based clustering algorithms iteratively reorganize the initial dataset into a set of partitions. The algorithms move objects at each iteration into different clusters while trying to optimize some criterion. Typical examples of partitioning-based clustering algorithms are: k-means, k-medoids or PAM (partitioning around medoids), k-medians, CLARA (clustering large applications), CLARANS (clustering large applications based on randomized search) and fuzzy c-means.

Many partitioning-based algorithms are variations of k-means. They divide objects into k clusters. Each cluster is represented by a center, which is calculated differently depending on the algorithm. In k-means the centroid is calculated as a mean of all points in the cluster, the k-medians algorithm uses medians and in k-medoids the centroid of a cluster is selected among objects, which are assigned to this cluster at the current iteration.

The k-means algorithm iteratively performs two steps:

- Each object is assigned to a cluster, which is the closest to the object. So the objective is to minimize the sum of Euclidean distances from the centers to the points in the corresponding clusters, i.e. the sum of the squared error (SSE) [26]:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2, \tag{2.10}$$

  where k is the number of clusters c_i - the center of ith cluster.

- The centers of the clusters are recalculated based on the objects assigned to them in the previous iteration:

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x, \tag{2.11}$$

where m_i is the number of objects assigned to a cluster i.

The k-means algorithm has the following advantages over others:

- the complexity of the algorithm is relatively low [26]:

$$O(I * k * m * n), \tag{2.12}$$

where I - number of iterations, that is required for the algorithm to converge, m - total number of objects, n - number of attributes (dimensions);

- it is simple and universally applicable.

However, it has drawbacks, which in some cases might be critical:

- sometimes k (the number of clusters) can be difficult to select;
- outliers and non-globular clusters are not handled properly by the k-means algorithm;
- results strongly depend on the initial centers selection strategy.

Another partitioning-based clustering algorithm which is widely used for database fragmentation is also a variation of k-means algorithm. It is called fuzzy c-means clustering. This algorithm allows a point to belong to more than one cluster. At each iteration instead of assigning a point to a cluster the algorithm calculates a membership degree (between 0 and 1) of the point to each cluster. The algorithm outperforms k-means algorithm in the case of an overlapping sets of objects.

CLARA - another example of partitioning-based clustering - is a modification of the PAM algorithm, which is commonly used for large datasets. The algorithm reduces an original dataset by selecting a random subset of objects from the original dataset. On the selected objects the classic PAM algorithm is performed. The algorithm runs multiple times and returns the best set of medoids found so far. Inspired by CLARA Raymond T. Ng and Jiawei Han proposed the CLARANS algorithm, which is based on randomized search.

Density-based clustering algorithms consider clusters as sets of objects in areas of high density. The algorithms of this class use the concepts of density, connectivity and

distances between objects and/or an object and a cluster. Such algorithms are very good at discovering clusters of arbitrary shapes and handling outliers [25]. Examples of density-based clustering algorithms are: DBSCAN (density-based spatial clustering of applications with noise), OPTICS (ordering points to identify the clustering structure), DenClue (density-based clustering) and HDBSCAN (hierarchical density-based spatial clustering of applications with noise).

DBSCAN is one of the most popular density-based clustering algorithms. The basic idea of the algorithm is to find areas of a higher density which are separated by areas of lower density [27]. Input values for the algorithm are: ε - neighborhood radius of a point and $minPts$ - minimum number of points in a neighborhood of a point x for x to be a core point. Core points, outliers and reachable points - these are the labels which can be assigned to a point by DBSCAN algorithm. A point a is reachable from a point b if there is a path between the points, which consists only of core points. If a point is not reachable from any other point in the set, it is called an outlier. Core points are those that have a given minimum number of points ($minPts$) within a given distance (ε). The DBSCAN algorithm finds all the core points first and merges [27] neighboring core points into clusters. Then it goes through the remaining points and assigns each point to a cluster if there is a core point which is within the radius ε from the point. Otherwise the point is considered to be an outlier. The DBSCAN clustering mechanism is visualized in Figure 2.10.

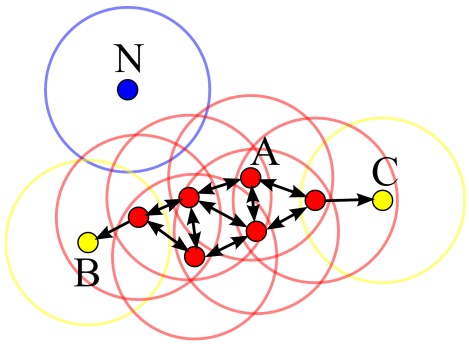


Figure 2.10: DBSCAN clustering mechanism [27]

Some advantages of DBSCAN include:

- there is no need to provide k - the number of clusters to form;
- as a density-based clustering algorithm it can discover clusters of different shapes;
- it handles outliers well.

Some disadvantages are:

- it is very hard to choose values for ε and $minPts$, especially if there are clusters of a different density;

- if there is a point, which is reachable from different clusters, it can be assigned to either of them.

Grid-based clustering algorithms reduce the object space by using a grid to divide the original dataset into groups and performing clustering taking grid cells as inputs. Usually the number of grid cells is much smaller than the number of data objects, which makes such algorithms run much faster. These algorithms perform very well on large multidimensional object spaces. The most commonly applied grid-based algorithms are [28]: Wave Cluster, STING (statistical information grid), CLIQUE (clustering in quest), O-CLUSTER (orthogonal partitioning clustering) and MAFIA (merging of adaptive intervals approach to spatial data mining). Sometimes grid-based clustering algorithms are considered to be a subclass of so-called subspace clustering algorithms. Subspace clustering algorithms in general are very good with clustering of high-dimensional data because besides clustering data objects itself they aim to select a subset of relevant dimensions (features) to perform clustering on.

The idea behind **model-based clustering algorithms** is that the data is generated by a mixture of probability distributions. Such methods are usually considered to be robust and flexible [25]. It is assumed that the objects of the same cluster will be part of the same distribution. Popular methods belonging to this group of clustering algorithms are: GMM (Gaussian mixture modeling) using EM (expectation–maximization algorithm), COBWEB (incremental system for hierarchical conceptual clustering) and SOM (self-organizing map).

Gaussian mixture modeling can be considered as a general case of k-means clustering. First, k Gaussian distributions are initialized, which correspond to k clusters. Each data object has a probability of belonging to each distribution. Parameters of the distributions are iteratively optimized and the probability of each object to belong to each distribution is re-estimated.

As shown in Figure 2.11, Gaussian mixture modeling can work much better than the classic k-means clustering algorithm if the data can be described by Gaussian distributions, but in a real world scenario this might not be the case.

Results of running some commonly used clustering algorithms on different input data are presented in Figure 2.12. The provided examples show how the choice of a clustering algorithm for a given dataset might influence the clustering results.

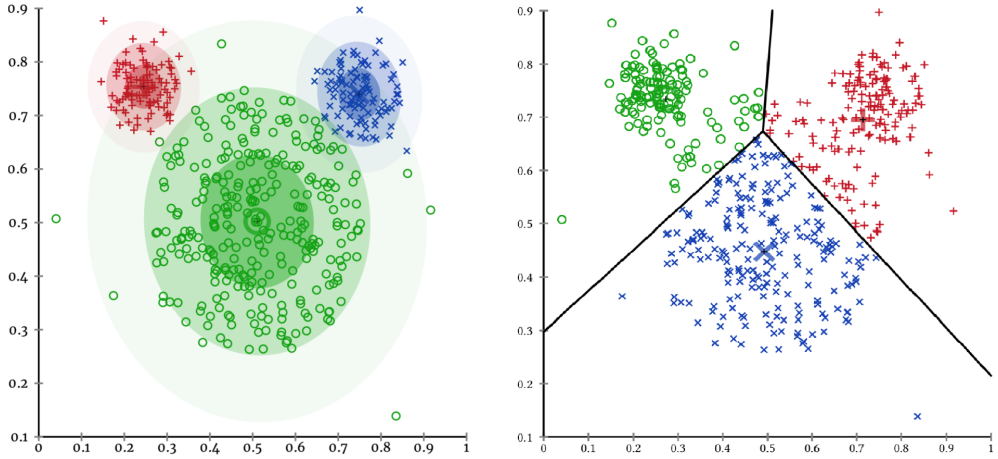


Figure 2.11: Results of clustering with Gaussian mixture modeling (a) and k-means (b) algorithms (k = 3) [29]

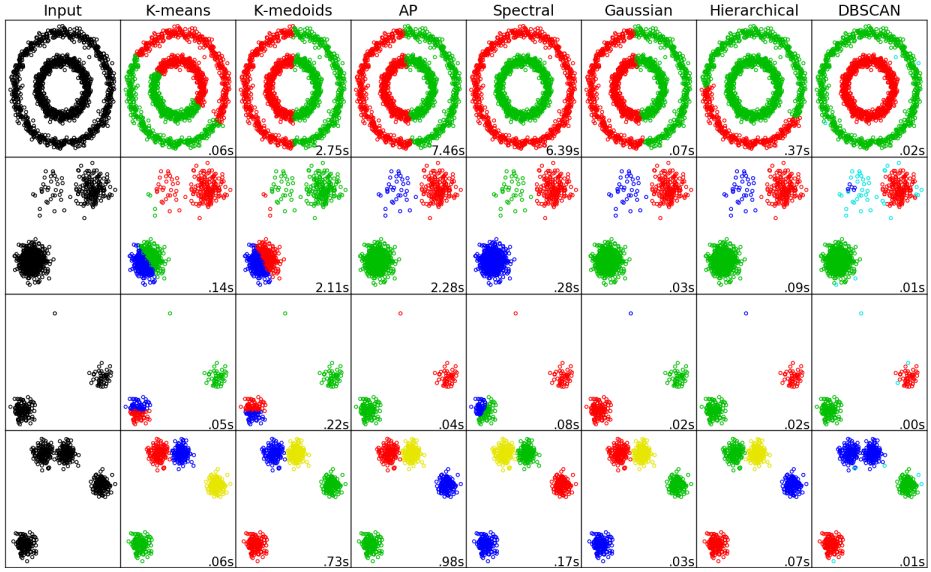


Figure 2.12: Results of clustering with commonly used clustering algorithms [30]

2.3.2.1 Hierarchical-based clustering

Another widely used type of clustering algorithms is hierarchical-based algorithms. Hierarchical-based clustering algorithms perform clustering in a hierarchical manner. Commonly used algorithms of this category are: DIANA (divisive analysis clustering), AGNES (agglomerative nesting clustering), BIRCH (balanced iterative reducing and clustering using hierarchies), CURE (clustering using representatives), ROCK (robust hierarchical clustering) and Chameleon.

Hierarchical-based clustering algorithms are pretty flexible and the developer can choose from a variety of implementation options and configuration parameters, such as: clustering strategy, similarity measure, linkage criteria and data structures used.

Considering clustering strategies, hierarchical clustering algorithms can be:

- Agglomerative (also known as a "bottom-up" strategy). Such algorithms start by assigning each data object to its own cluster, and then they iteratively merge the two closest clusters into a single cluster. Once a pair of clusters is merged it can never be split.
- Divisive (also known as a "top-down" strategy). Divisive hierarchical clustering algorithms start by moving all data points into a single cluster, and then they iteratively split a cluster from the set of clusters obtained so far. Once a cluster is split into two parts these parts can never be merged again.

Agglomerative and divisive strategies of hierarchical clustering are presented in Figure 2.13.

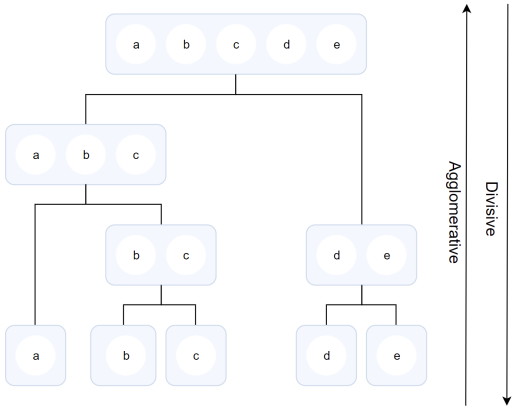


Figure 2.13: Agglomerative and divisive hierarchical clustering

Such split/merge operations and their order can be visualized using a dendrogram. The dendrogram also contains information about how similar the clusters are to each other - the y-axis represents the degree of dissimilarity between the clusters (Figure 2.14).

There are several approaches that can be used to extract clusters from a dendrogram [29]:

- If k is specified, then a horizontal line should be drawn so that the number of intersecting vertical lines is equal to k (Figure 2.14). Clusters that are formed below this line are the output of the algorithm.
- If k is not specified, the clustering process can be stopped once certain size or density of the clusters is reached. The clustering process can also terminate once some application-specific constraints are satisfied.

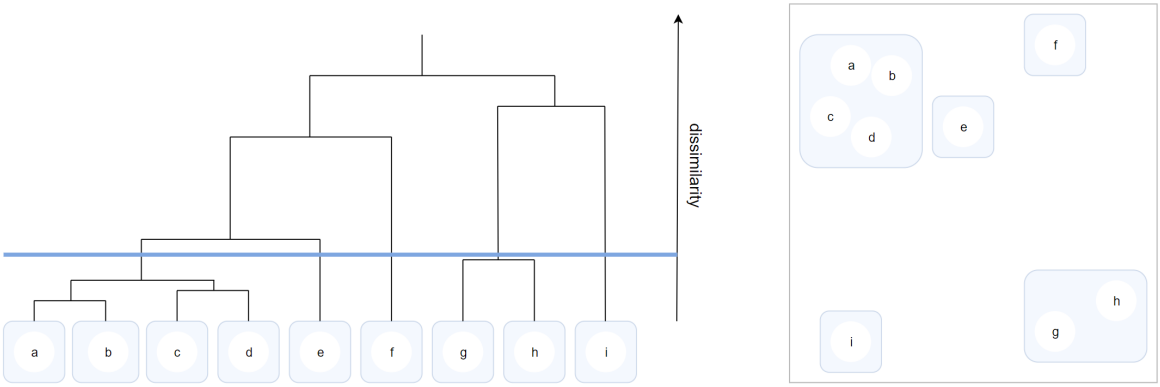


Figure 2.14: Dendrogram and clusters generated for k = 5

The time complexity of the classical agglomerative hierarchical clustering is $O(n^3)$. It can be reduced to $O(n^2\ log\ n)$ by using a heap for storing distances between each pair of clusters. As for the divisive hierarchical clustering, there are 2^n possible splits, that is why heuristic approaches should be used to reduce the time complexity [29].

One undoubted advantage of the hierarchical-based clustering algorithms is configurability, given that a variety of similarity measures and linkage criteria can be used.

Linkage criteria is basically a function, which determines how distance between two clusters is calculated. There are multiple linkage criteria that are often used with hierarchical clustering:

- **Minimum** or **Single-linkage**.

  This linkage criterion calculates distance between two clusters as a distance between two of their closest points (Figure 2.15). As pointed in [31], Single-linkage criterion may lead the algorithm to form clusters, which are generated by individuals iteratively added to the same group.

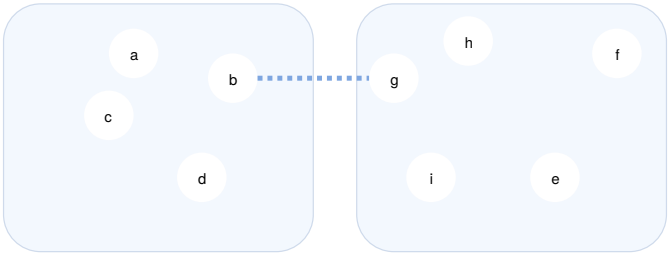


Figure 2.15: Single-linkage criterion

- **Maximum** or **Complete-linkage**.

  This linkage criterion calculates the distance between two clusters as a distance between two of their farthest points (Figure 2.16). Clusters, which are generated using this linkage criterion are usually well split and compact [31].

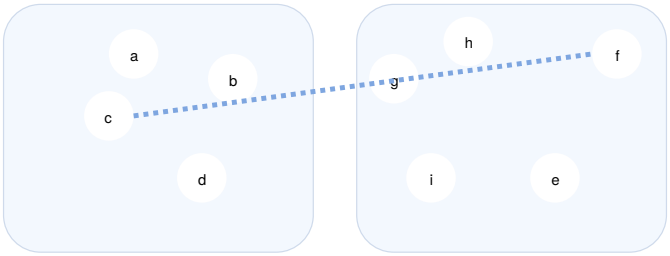


Figure 2.16: Complete-linkage criterion

- **Average-linkage**.

  Using this approach the distance between two clusters is calculated as the average distance between all pairs of points from different clusters (Figure 2.17).

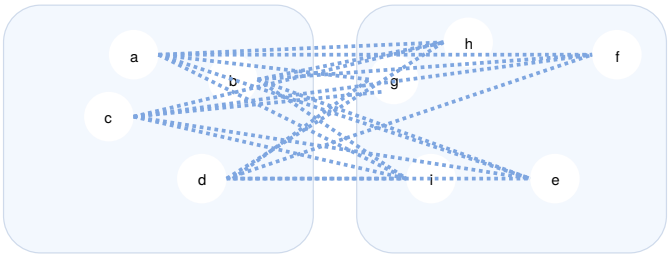


Figure 2.17: Average-linkage criterion

- **Centroid-linkage**.

  The distance between two clusters is calculated as the distance between the centroids of the clusters (Figure 2.18). This linkage criterion can only be used in combination with the Euclidean distance similarity measure.

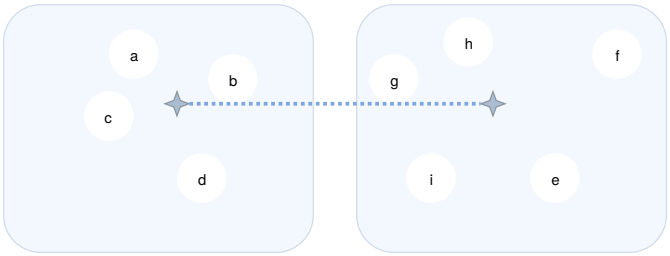


Figure 2.18: Centroid-linkage criterion

- Other linkage criteria exist, including weighted average-linkage, minimum energy linkage, median-linkage, Ward linkage, Min-Max linkage.

Similarity measures show how similar two data objects are. There are multiple similarity measures that can be used in hierarchical clustering algorithms. Some of them are:

- **Euclidean distance**. This is probably the most widely used similarity measure. It calculates the straight-line distance between two data points:

$$d_e(x, y) = \sqrt{\sum_{d=1}^{n}(x_d - y_d)^2} \tag{2.13}$$

- **Squared Euclidean distance** is calculated as follows:

$$d_e^2(x, y) = \sum_{d=1}^{n}(x_d - y_d)^2 \tag{2.14}$$

- **Manhattan distance** (also known as city-block distance):

$$d_m(x, y) = \sum_{d=1}^{n}|x_d - y_d| \tag{2.15}$$

- **Maximum distance** (along with Euclidean and Manhattan distances) is a special case of Minkowski norm [29] and can be calculated as follows:

$$d_{max}(x, y) = max_d|x_d - y_d| \tag{2.16}$$

It should be noted that the terms "distance", "metric" and "similarity" are often used interchangeably, but from a mathematical point of view they are different [29].

Apart from the above-mentioned similarity measures and linkage criteria, an expert can create his own application-specific linkage criteria and similarity measures, based on his knowledge about the application domain.

Hierarchical-based clustering algorithms provide very good clustering results, especially for small datasets. As mentioned before, they are very flexible, which allows them to be configured to pretty much every dataset. Such algorithms are transparent and their results are easy to interpret. Moreover, unlike partitioning-based algorithms, these algorithms do not require the number of clusters to be specified beforehand.

However, hierarchical-based clustering algorithms might not be the best choice for a large dataset, considering their computational requirements. Another problem is that the changes made once cannot be undone, which can lead to suboptimal results. Such algorithms also do not use a global optimization function. Furthermore, hierarchical-based clustering algorithms often do not handle outliers properly [29].

2.3.2.2 Clustering-based horizontal fragmentation

In the last century almost all presented horizontal fragmentation algorithms were purely heuristic. Nowadays many of them are based on machine learning techniques and clustering algorithms in particular. These relatively new approaches seem to be a promising research direction, since machine learning solutions are considered to be more flexible and adjustable than complex tailored heuristics.

In this chapter the results of a thorough literature review on applications of clustering in horizontal fragmentation are presented. There are numerous studies devoted to this topic, but probably the most extensive research in this direction with object-oriented databases was conducted by the scientific group of Adrian Darabant.

In their work [32] Adrian Darabant and Alina Câmpan perform horizontal fragmentation for an OO database, using the k-means clustering algorithm. Clusters are formed based on the similarity between class instances, which is calculated using Cosine and Manhattan similarity measures. Authors state that different similarity measures as well as different methods for choosing the initial cluster centroids have a huge impact on the fragmentation results. Although clustering using Manhattan distance in general outperforms the one using Cosine similarity, neither of them is shown to perform optimally in all cases.

In their study [33] authors use the same input data representations and similarity measures but, in contrast to their previous work, authors use agglomerative hierarchical clustering. In each iteration the algorithm merges two closest (according to the selected similarity measure) clusters. Despite very promising results authors note that the reason why the proposed approach does not always produce the optimal result lies in the hierarchical clustering algorithm itself - the algorithm cannot revert erroneous decisions made once.

The k-means-based and the hierarchical-based fragmentation approaches from the papers discussed above were compared in a subsequent work [34].

In 2009 [35] authors conducted extensive experiments on the effectiveness of hierarchical clustering applied to the horizontal fragmentation problem. They compared fragmentation scheme generated by it with the one generated by the k-means algorithm over baselines of single site and fully replicated databases. Authors also compared the results with the fragmentation schemes generated using algorithms proposed by Baiao et al. [36] and Bellatreche [37]. The results show that hierarchical clustering with the object-condition matrix as input data representation and Manhattan similarity measure outperforms both k-means clustering and the baselines. Hence, this work establishes that hierarchical clustering algorithm can indeed be successfully applied to the problem. Moreover, the results corroborate that different similarity measures and input representations can drastically affect the clustering results. However, it is worthwhile to mention that with the growth of the database k-means algorithm starts performing better than hierarchical clustering.

Another paper called "Hierarchical Clustering in Object Oriented Data Models with Complex Class Relationships" [38] extends their previous work [33] by taking into account aggregation and association relationships between classes. In this way authors perform primary horizontal fragmentation and derived horizontal fragmentation. Authors introduce two new input representations, namely extended object-condition matrix (EOCM) and extended characteristic matrix (ECVM), and use them as inputs for agglomerative hierarchical clustering. In follow-up works [39, 40] authors apply k-means clustering algorithm to the same problem.

Our research in clustering-based horizontal fragmentation was partially inspired by two papers from the same team of authors [41, 42]. These papers analyse the influence of different similarity measures on the quality of fragmentation results obtained using agglomerative clustering. Although authors consider an object-oriented data model (including complex class hierarchies) and use different input data representations, the obtained results are highly relevant to the research in this thesis. Three similarity measures are considered in the paper: Cosine, Euclidean and Manhattan. According to the study, the best results are achieved by using the Euclidean similarity measure, followed by the Manhattan similarity measure. Both measures in general outperform Cosine similarity. For primary horizontal fragmentation the results obtained using different similarity measures do not differ much though. In our thesis we would like to test how the choice of a similarity measure will influence the results of horizontal fragmentation in a relational database obtained using other input data representation. We would like to consider other similarity measures as well, such as Maximum distance or squared Euclidean distance.

In related work [43] authors use fuzzy c-means clustering algorithm to perform horizontal fragmentation and replication in one step. Fuzzy c-means algorithm assigns an object to one or multiple clusters and this feature is adapted to handle the replication task. Fuzzy c-means generates a probability for each object to belong to a cluster. Unlike traditional clustering algorithms, such as k-means or hierarchical clustering, fuzzy c-means clustering takes into account the fuzzy nature of the fragmentation process and allows fine-grained replication to be integrated into it. This is the reason why the proposed algorithm outperforms previously developed k-means-based clustering approaches.

A paper published in 2011 [44] summarizes the research done by Adrian Sergiu Darabant and his research group in previous years. Authors conduct experiments with hierarchical, k-means and fuzzy c-means clustering algorithms applied to the horizontal fragmentation task. They compare algorithms with fully replicated and centralized databases along with the *Min_Complete* algorithm proposed by Ezeife and Barker [45].

Information on the papers written by Adrian Darabant and his scientific group is summarized in Table 2.6.

The team of Darabant et al. is not the only group that is working in this direction. Other researchers solve the same problem in different environments, using different clustering algorithms and input data representations.

Paper	Year	Input data representation	Clustering algorithm	Similarity measure / Linkage criterion
"Semi-supervised learning techniques: k-means clustering in OODB Fragmentation"	2004	object-condition matrix, characteristic vector matrix	k-means	Cosine, Manhattan
"AI clustering techniques: a new approach in horizontal fragmentation of classes with complex attributes and methods in object oriented databases" / "A new approach in fragmentation of distributed object oriented databases using clustering techniques"	2004 / 2005	extended object-condition matrix, extended characteristic matrix	k-means	Euclidian, Manhattan
"AI Clustering Techniques: a New Approach to Object Oriented Database Fragmentation"	2004	object-condition matrix, characteristic vector matrix	agglomerative hierarchical clustering	Cosine, Manhattan + average linkage
"Hierarchical clustering in object oriented data models with complex class relationships"	2004	extended object-condition matrix, extended characteristic matrix	agglomerative hierarchical clustering	Cosine, Manhattan + average linkage
"Advanced Object Database Design Techniques"	2004	object-condition matrix, characteristic vector matrix	k-means, agglomerative hierarchical clustering	Cosine, Manhattan

"Using Fuzzy Clustering for Advanced OODB Horizontal Fragmentation with Fine-Grained Replication"	2005	extended object-condition matrix	Fuzzy c-means	Euclidian, Manhattan
"The similarity measures and their impact on OODB fragmentation using hierarchical clustering algorithms" / "A comparative study on the influence of similarity measures in hierarchical clustering in complex distributed object-oriented databases"	2006	extended object-condition matrix, extended characteristic matrix	agglomerative hierarchical clustering	Euclidian, Manhattan, Cosine
"Hierarchical clustering in large object datasets — a study on complexity, quality and scalability"	2009	object-condition matrix, characteristic vector matrix	agglomerative hierarchical clustering	Euclidian, Manhattan
"Clustering methods in data fragmentation"	2011	extended object-condition matrix, extended characteristic matrix	agglomerative hierarchical, k-means and fuzzy c-means clustering	Euclidean, Manhattan + average linkage for hierarchical clustering

Table 2.6: Clustering in horizontal fragmentation 1

Recently clustering algorithms were applied to horizontal fragmentation in XML data warehouses [46, 47]. Authors claim that the control over a number of horizontal fragments is a very important issue in data warehouses; therefore, a horizontal fragmentation algorithm based on k-means clustering was used in the paper. The algorithm takes an XML data warehouse and a corresponding query-workload as inputs. The predicates from the workload are grouped using the clustering algorithm. The proposed algorithm is compared with predicate-based and affinity-based fragmentation algorithms adapted from the relational domain. Authors compare query response time and fragmentation costs for the algorithms. Although the proposed algorithm outperforms classical fragmentation algorithms, it is worthwhile to mention that it is because of the inability of the classical algorithms to control the number of generated fragments. Therefore, it is not clear whether the proposed algorithm would provide better results if the number of fragments was the same.

Agglomerative hierarchical clustering was also used for horizontal fragmentation in multimedia databases [48]. Authors use selectivity of predicates and access frequencies of the queries to perform fragmentation. PUM (predicate usage matrix) was selected as input data representation. In our opinion, the main disadvantage of the proposed approach is using the Single-linkage for hierarchical clustering. That means that if two clusters were merged, in the next iteration the algorithm only takes into account similarity between the closest points, which belong to the clusters. Moreover, clustering of the predicate usage matrix does not guarantee two correctness properties of a fragmentation: reconstructability and disjointness.

An interesting perspective on hybridized fragmentation problem was introduced by Harikumar et al. [49]. Authors try to find some patterns in the data to guide the hybridized fragmentation process. Using subspace clustering they identify correlated attributes for a set of tuples. Projected clustering (a type of subspace clustering) helps to find these patterns in the subspaces of high-dimensional data. Then the clusters formed by projected clustering are merged to obtain a specified number of fragments. Clusters are merged using hierarchical clustering based on Jaccard distance. Authors use the execution time of the queries as a cost model for evaluation. The databases were connected with *dblink* and it takes quite a long time to establish a remote connection. This might be the reason why the proposed algorithm is sometimes outperformed by a random fragmentation.

Another application of clustering algorithms for fragmentation is demonstrated in the paper of Luong et al. [50, 51]. Authors use Knowledge-Oriented clustering algorithm proposed by Shoji Hirano and Shusaku Tsumoto [52] both for vertical and horizontal fragmentation. The proposed algorithm takes a tuple-predicate matrix as input, and the similarity between two objects is calculated using Jaccard coefficient. Results obtained by the algorithm are similar to the results obtained by classical fragmentation algorithms, but unfortunately the generated fragmentation scheme was not evaluated in the papers and we cannot analyze its quality.

In 2015 a clustering-based algorithm for horizontal fragmentation of real-time data warehouses was proposed [53]. Authors propose an approach that consists of two levels (phases). The first phase is initial fragmentation using Gaussian-means clustering. During this phase an optimal amount of clusters is determined. The second phase was added to balance the amount of tuples in the fragments by merging and dividing existing fragments when new data arrives to the system. In the context of this thesis the first phase is the most interesting. On the one hand, the proposed clustering algorithm based on Gaussian distribution does not require a number of fragments as an input parameter, which makes it more flexible. On the other hand, it requires a standard statistical significance level α to be specified in advance. Moreover, the chosen input representation (predicate usage matrix) does not guarantee that the generated fragmentation scheme conforms to the correctness properties of reconstructability and disjointness.

A clustering technique for horizontal fragmentation of fuzzy object-oriented databases was recently introduced by Nguyen et al. [54]. Fuzzy object-oriented databases combine the object-oriented database model with fuzzy mathematics to represent attributes with uncertain values. To find fuzzy clusters in quantitative dataset authors use statistical theory. They apply expectation maximization coefficient clustering (an improvement over the expectation maximization algorithm) that uses the coefficient of variation in order to increase the softness level in the clustering. This coefficient determines the density distribution of the elements in a cluster. The results generated by the clustering algorithm are then used for identifying fuzzy intervals of a quantitative attribute.

Another approach was studied by Sun in the context of relational database management systems [55]. Here, data is grouped into blocks, each containing metadata that describes the data inside the block. That allows query engines to know whether there is data to read inside a block. This forms the idea of *aggressive data skipping*, a feature of data systems where data is fragmented both horizontally and vertically, such that the number of blocks that can be skipped is maximized. Author develops three systems that build on each other: SOP (skipping-oriented partitioning), GSOP (generalized SOP) and GSOP-R (GSOP with replication). SOP establishes the core technique of feature-driven workload analysis. Through this approach, the queries that constitute the expected workload (for which the fragmentation is optimized) are analyzed and reduced to a small number of succinct predicates which subsume most queries in the workload. Then tuples are finally distributed into blocks in such a way that each block is represented by a single feature vector (called union vector, formed by a logical OR of the feature vectors contained). If there is a 0 for a given predicate in the union vector, then no tuple inside the block satisfies the predicate, otherwise there might be tuples in the block that satisfy it. Author defines the optimization goal of achieving the maximum skip partitioning and solves this optimization problem using hierarchical clustering. Unfortunately, no comparison of the proposed approach with other classical fragmentation algorithms was presented.

The above-mentioned papers are summarized in Table 2.7.

Paper	Year	Authors	Input data representation	Clustering algorithm	Similarity measure / Linkage criterion	Database model
"Data Mining-based Fragmentation of XML Data Warehouses" / "Fragmenting very large XML data warehouses via K-means clustering algorithm"	2008 / 2017	Alfredo Cuzzocrea, Jérôme Darmont, Hadj Mahboubi	query-predicate matrix	k-means	Euclidean	XML
"Horizontal Partitioning of Multimedia Databases Using Hierarchical Agglomerative Clustering"	2014	Lisbeth Rodríguez-Mazahua, Giner Alor-Hernández, María Antonieta Abud-Figueroa, Silvestre Gustavo Peláez-Camarena	predicate usage matrix	hierarchical clustering	Euclidian + Single-linkage	Multimedia
"Hybridized Fragmentation of Very Large Databases Using Clustering"	2015	Sandhya Harikumar, Raji Ramachandran	a point (feature vector) of a cluster is a record and a dimension (feature) is an attribute.	subspace clustering (Proclus) + hierarchical clustering	Manhattan	Relational

"An improvement on fragmentation in Distribution Database Design Based on Knowledge-Oriented Clustering Techniques" / "An improvement on fragmentation in Distribution Database Design Based on Clustering Techniques"	2015	Van Nghia Luong, Ha Huy Cuong Nguyen, Van Son Le	tuple-predicate matrix	Knowledge-Oriented Clustering Technique Based on Rough Sets	Jaccard coefficient	Relational
"2LPA-RTDW: a two-level data partitioning approach for real-time data warehouse"	2015	Issam Hamdi, Emna Bouazizi, Saleh Alshomrani, Jamel Feki	predicate usage matrix	Gaussian-means	-	Relational
"Skipping-oriented Data Design for Large-Scale Analytics"	2017	Liwen Sun	feature vectors (based on simple predicates)	hierarchical clustering	Ward's criterion	Relational
"Clustering and Query Optimization in Fuzzy Object-Oriented Database"	2019	Thuan Tan Nguyen, Ban Van Doan, Chau Ngoc Truong, Trinh Thi Thuy Tran	-	expectation maximization coefficient clustering	-	Fuzzy OO

Table 2.7: Clustering in horizontal fragmentation 2

2.3.3 Reinforcement learning

Reinforcement learning(RL) is a type of machine learning defined usually by its problem: learning from iterative interaction with an external system to achieve a goal. The external system signals the learner about the current state it is in and what is the reward of reaching this state. The learning model is then tasked to pick an action to try to maximize the long-term total reward. The RL framework consists of several entities:

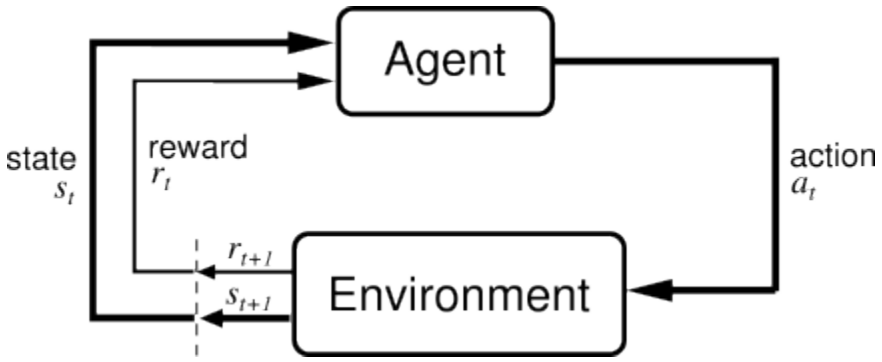


Figure 2.19: Agent-environment interaction in RL [56]

- **Agent**: a model that learns and makes decisions regarding which action to take given a state. This is the "learning" part of RL.
- **Environment**: the system the agent interacts with. Generally, it is an external application-specific concept, which represents the domain the RL algorithm needs to navigate.
- **State**: a signal from the environment relaying to the agent its current position. It is denoted as $s_t \in S$, where S is a finite *state space*.
- **Action**: a change of state as a result of a decision made by agent. It is denoted as $a_t \in A$ where A is finite *action space*. In this work we assume that A does not change with state s_t.
- **Reward**: the reinforcement signal from environment to agent, relaying to agent information about the quality of the state it reached. The goal of the agent is to maximize the total rewards from reached states.
- **Step**: single discrete moment of interaction between agent and environment. State, actions and reward are all measured for specific steps.
- **Episode**: single instance of agent-environment interaction process. Typically in RL the number of steps for single learning/interaction is limited either by some internal environment constraint or, in case of indefinite task optimization, by the training regime.

In its more formal specification, RL is defined as a task of learning the policy for a Markov decision process (MDP) which maximizes the total reward. A specific MDP is defined by its state and action spaces, as well as its single-step dynamics. Each particular finite MDP is defined in terms of its *transitional probabilities* [56]:

$$P^a_{s,s'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.17}$$

for each action a and state s. These define the probability to transition to state s_{t+1} from state s_t when performing action a_t. The estimated reward for each state and action is denoted as [56]:

$$R^a_{s,s'} = E\{r_{t+1} | s_t = s, a_t = a\} \tag{2.18}$$

The agent is tasked with learning a *policy*: a mapping function between the current state and action that needs to be taken. The policy of an agent can either be deterministic (i.e. every state determines the exact action that needs to be taken) [56]:

$$\pi(s_t) = a_t; s_t \in S, a_t \in A \tag{2.19}$$

or it can be stochastic, where each action is assigned a certain probability [56]:

$$\pi(a_t | s_t) = p_i; s_t \in S, a_t \in A, 0 \leq p_i \leq 1 \tag{2.20}$$

2.3.3.1 Basic RL

To tackle the task of accruing the most reward during an episode, an agent needs to store and improve its understanding of the environment dynamics. There are some basic methods that allow agents to generalize such information. Besides direct policy representation, where the policy that the agent refines is stored in the way presented above, there exist the following approaches:

- **Value function:** is a function of a state which estimates how good the state is for an agent's goal. This includes the agent's estimation of the rewards for future steps it is going to take. Given the agent following policy π, we can denote its value function as follows [56]:

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \tag{2.21}$$

  Using the Bellman equation, the agent can iteratively improve the precision of its value function through dynamic programming methods [56]:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \tag{2.22}$$

- **Action-value function:** similarly to value functions, the action-value function defines the quality of taking action a on step s when following the policy π [56]:

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s, a_t = t\} \tag{2.23}$$

Since the task of the RL agent is to maximize the total, we can rate the agent's policies using value and action-value functions. If the policy π is better than policy π', its expected returns should also be greater (i.e. $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in S$). One can define the the *optimal policy* π^*, which is better or equal to all other policies. Then, this policy has to have an *optimal value function* [56]:

$$V^{\pi^*}(s) = max_{\pi} V^{\pi}(s) \tag{2.24}$$

and *optimal action-value function* [56]:

$$Q^{\pi^*}(s,a) = max_{\pi} Q^{\pi}(s,a) \tag{2.25}$$

Policy iteration

Both value and action-value function can be computed and improved using the Bellman equation. Starting with some policy π for which we have calculated $V^{\pi}(s)$ and $Q(s, \pi(s))$, we can try to replace it with policy π'. If the policy π' is different only in one step s_t, we can make sure that the new policy is better than π by checking the expected returns of both policies [56].

Dynamic programming

At the core of iteratively improving the target policy lies the concept of *greedy policy*. It changes the target policy by looking ahead one step and using the target policy for value estimation from there, and then picking the action that has a highest estimated reward [56]:

$$\pi'(s) = argmax_a \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{2.26}$$

Policy improvement in such way always yields better policies, only excluding the cases where the target policy is already optimal. Once the policy was improved, the resulting policy π' can be used in the next iteration as a target policy, resulting in a sequence of monotonically improving policies.

However, such approach requires multiple lookups through the state set to recalculate the new policy's value function. This can be avoided by using fixed-step look-ahead

and value function backups, shortening a number of steps needed to evaluate the value function. The single-step look-ahead variant of such approach is called *value iteration*.

Monte Carlo methods

Monte Carlo (MC) methods represent an elementary solution or RL problem where the agent does not assume the full knowledge of the environment. Instead, MC methods rely on learning only from the interaction with the environment they have experienced in the past. Typically, the MC method is used to estimate the value or action-value functions by averaging the returns for each state (or state-action) for each episode in the training set.

MC methods borrow the notion of greedy policy, using it as a resulting policy once the estimation of the action-value function gets good enough. However, the drawback of such approach is that it is impossible to use the target policy for training runs. If the greedy policy would be used as a navigation guide for training, it would prevent the learning by choosing only the known paths, avoiding exploration. This leads to the classic problem of exploration vs exploitation, where following a greedy policy may cause worse results *in the future*, but might improve the reward *now*. It makes sense to explore abundantly, as the agent learns from the environment, but let the agent focus more when it has sufficient knowledge about the value of actions.

There are multiple ways of managing this contradiction, some of them are:

- The **ϵ-greedy** approach proposes to base the policy selection of an agent on some variable ϵ which changes during the course of the training. This value represents the probability of using the training strategy (which most often means random choice of action) at each step. In most cases, ϵ is higher in earlier phases of training, decaying over time to focus search and improve the total reward. There are various applications of this approach with different functions of ϵ-decay.

- **Boltzmann exploration** is an approach designed to alleviate the hard switching between greedy and random policies of ϵ-greedy. The idea here is that the training choice of ϵ-greedy does not depend on the accumulated experience, which might be a drawback in the later stages of learning. To use the experience when following training policy it is proposed to assign each action of the state a probability using Boltzmann distribution:

$$p(a_t|s_t, Q_t) = \frac{e^{Q(s_t,a_t)/\tau}}{\sum_{b \in A_s} e^{Q(s_t,a_t)/\tau}} \tag{2.27}$$

  The τ parameter here is used to control the training, the lower it is the greedier is policy's choice. One can apply decay functions to τ mentioned earlier to facilitate proper training regime for MC and MC-based methods.

Temporal-difference learning

The idea behind temporal-difference (TD) learning is that of combining policy bootstrapping of dynamic programming (DP) with raw experience learning of MC methods. The key idea here is using experience of previously visited states to update value function estimation, thus making better estimations in the future. TD brings the benefit of DP in that the target policy can be used in estimation, allowing the MC approach to estimate before the end of an episode. TD improves upon DP in that it does not require the model of the environment to quickly calculate full backup of the value function. TD also has an advantage over MC in that it allows for fully iterative online learning. Typically, TD algorithms provide better results than purely DP and MC methods, especially on unpredictable or unknown environments [56].

There exist two major types of approaches to TD in terms of action-value function estimation:

- **On-policy** methods use the target policy to update estimations of $Q(s, a)$. An example of such method can be SARSA algorithm [56].
- **Off-policy** methods use policies different from the target policy to update estimations of $Q(s, a)$. An example of such method can be Q-learning [57].

SARSA

State–action–reward–state–action (SARSA) is an algorithm which employs on-policy TD control. In contrast with previously discussed methods, SARSA algorithm estimates action-value function instead of state-value function, updating its estimation each step. The goal is to build a $Q(s, a)$ table that is used by ϵ-greedy or ϵ-soft policy, using the same policy to traverse the training episodes. In short, the SARSA algorithm can be depicted as follows [56]:

Algorithm 3: SARSA

$Q(s, a) \leftarrow$ initial estimation
for each episode **do**
  $s \leftarrow START$
  $a \leftarrow \pi(s)$ {e.g. π is ϵ-greedy or ϵ-soft}
  repeat
    $s', r \leftarrow step(s, a)$
    $a' \leftarrow \pi(s')$ {e.g. π is ϵ-greedy or ϵ-soft}
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a'$
  until s is terminal
end for

Q-learning

Q-learning is another TD algorithm for RL, which uses off-policy control. Its procedural representation is shown in Algorithm 4:

Algorithm 4: Q-learning

$Q(s, a) \leftarrow$ initial estimation
for each episode **do**
  $s \leftarrow START$
  repeat
    $a \leftarrow \pi(s')$ {π is policy derived from Q, e.g. ϵ-greedy or ϵ-soft}
    $s', r \leftarrow step(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  until s is terminal
end for

The most prominent feature of Q-learning is that the estimated action-value function directly approximates an optimal action-value function regardless of the target policy. The policy still plays its role determining which state-action pairs to visit, but the only requirement of convergence is that all pairs continue being updated [56].

For its simplicity and ease of analysis combined with good convergence, Q-learning is used as a base for a number of more sophisticated algorithms, discussed below.

Function approximation

The assumed format of storing value and action-value functions is that of a table, mapping s and s, a to some value. Considering huge state- and action-spaces or continuous states or actions such format of storage does not scale well. Creation, storage and update of such data structures quickly becomes the main bottleneck of an RL solution.

One of the possible solutions to this problem is applying existing techniques of *function approximation*. This means that instead of storing the table for action-value function, we represent it in parametrized functional form with parameter vector $\vec{\theta_t}$. Typically, the number of items in $\vec{\theta_t}$ is much smaller than the number of states or state-action pairs, thus allowing to compress the data needed to store the function approximation. This, however, means that instead of of $Q(s, a)$ we would instead learn some function that which approximates $Q(s, a)$:

$$Q^*(s, a, \theta) \approx Q(s, a) \tag{2.28}$$

Given the nature of lossy compression of function approximators, update on an approximated function by some actual reward from an environment for some particular action-state pair actually affects many other states. The approaches to function approximation are typically borrowed from supervised learning, with tweaks that allow for step-by-step update via bootstrapping.

That said, for practical applications function approximation serves as a reasonable approach to use in RL algorithms. There exist multiple algorithms which employ various kinds of multivariate regression, decision trees or neural networks to approximate action-value function.

The task of a function approximator in an RL solution is to bring the parametrized function towards the target Q. A key metric of approximation accuracy is mean squared error:

$$L(\theta) = \frac{1}{N} \sum_{i \in N} [Q(s_i, a_i) - Q_\theta(s_i, a_i)]^2 \tag{2.29}$$

The most popular class of methods of learning for function approximation are gradient-descent based methods. The idea of gradient descent approach is moving the $\vec{\theta_t}$ in the direction of the largest possible decrease in error for $Q_{theta}(s, a)$, which is determined by its gradient. If we assume that we have a number of training observations $s_t, a_t \mapsto Q(s_t, a_t)$ on step t, the equation to update the $\vec{\theta_t}$ values looks as follows:

$$\vec{\theta_{t+1}} = \vec{\theta_t} - \frac{1}{2} \alpha \nabla_{\vec{\theta_t}} [Q(s_t, a_t) - Q_\theta(s_t, a_t)]^2 \tag{2.30}$$

The variable α is a step-size parameter. It serves the purpose of limiting change of $\vec{\theta_{t+1}}$, as overfitting the approximation on one state can lead to loss of accuracy on other steps. The intention is to reduce α throughout training to improve the stability of the approximator.

In practical applications, calculating and storing actual $Q(s, a)$ for all t is also infeasible. To alleviate that one can use bootstrapping with λ-steps lookahead like described earlier. Parameter update equation for single-step lookahead algorithm like Q-learning looks as follows:

$$\vec{\theta_{t+1}} = \vec{\theta_t} - \frac{1}{2} \alpha \nabla_{\vec{\theta_t}} [r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)]^2 \tag{2.31}$$

2.3.3.2 Deep RL

One of the most prominent state-of-the-art approaches in function approximation is that of deep learning. With an advent of fast and easy-to-deploy neural network implementations, deep neural networks (DNNs) have become a viable solution in multiple domains, such as computer vision, pattern recognition, speech recognition, natural language processing, and recommendation systems [58]. Deep learning is an umbrella term for all applications of DNNs for machine learning tasks.

DNN in RL not only serves the purpose of function approximation, but also allows more freedom for state and action space representations. With the developments of different

neuron types (convolutional, recurrent, GANs) the input data profile of the DNN fits pretty much any application.

In Figure 2.20 we can see the taxonomy of recent RL algorithms. There is a major division into model-free and model-based RL, from which we chose to focus on model-free class of algorithms. A further divide splits model-free approaches into policy optimization and value-based methods.

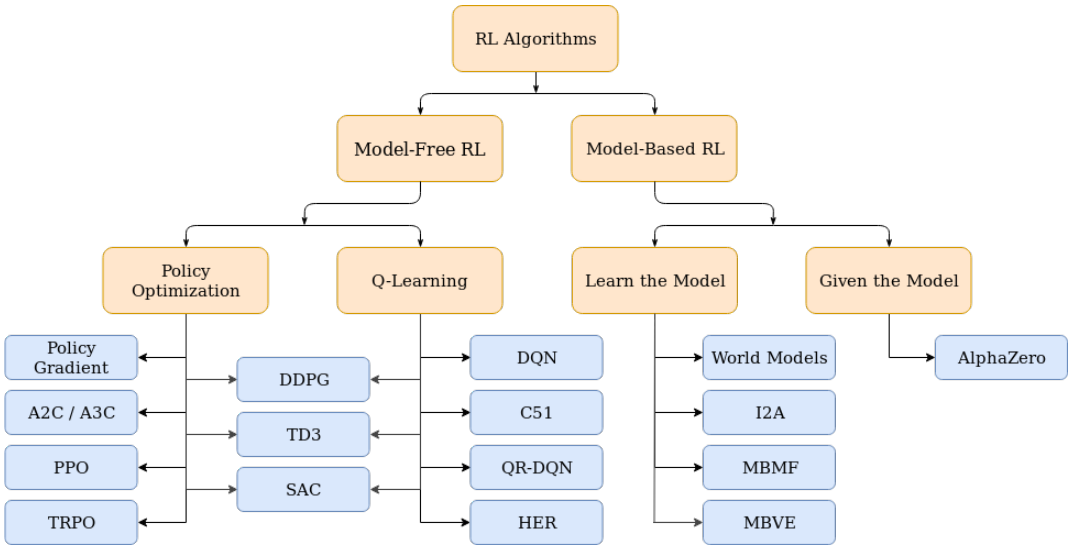


Figure 2.20: Taxonomy of modern RL algorithms[1]

Deep Q-Network

Deep Q-Network is an algorithm proposed by Mnih et al., which uses DNN in a variant of Q-learning algorithm to learn policies from high-dimensional input [59]. This particular agent was learning how to play Atari games from raw bitmap screen representations.

In the work of Mnih et al., $Q(s, a)$ is substituted by a convolutional neural network. The inputs of the neural network are pixels of Atari 2600 video output, outputs are a vector Q which represents action-value per each available action (this particular algorithm assumes that all actions are available and are the same at each state). Given state s_i, the neural network produces approximated $Q(s, a)$ for each action at once.

Training is performed by minimization of the loss function at training iteration i:

$$L_i(\theta_i) = E_{(s_t,a_t,s_{t+1},r_{t+1})r\sim D}[r_{t+1} + \gamma \max_{a_{t+1}} \overbrace{Q(s_{t+1}, a_{t+1}; \theta_i)}^{\text{target network}} - \underbrace{Q(s_t, a_t; \theta_t)}_{\text{current network}}]^2 \tag{2.32}$$

Using the neural network to calculate both the target function and current function leads to convergence instability: during the same episode changes in target and current

[1] https://spinningup.openai.com

functions develop a strong correlation. To alleviate that, DQN proposes to separate target and current networks, fixing the target weights for the whole episode (or even multiple episodes) while current continuously updates.

Another improvement of DQN over naive Q-learning with neural network is the introduction of so called *experience replay* to improve convergence. The training samples $(s_t, a_t, s_{t+1}, r_{t+1})$ are stored in a special buffer D, from which they are sampled in batches to train the network. This helps to propagate new and "controversial" experience through the function approximator. In DQN the batches are selected uniformly.

The algorithm of DQN training is presented in Algorithm 5:

Algorithm 5: Deep Q-learning with Experience Replay [59]

Initialize D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for t = 1, T **do**
    With probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    $$y_j \leftarrow \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a_j; \theta), & \text{for non-terminal } \phi_{j+1} \end{cases}$$
    Perform gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Improvements over DQN

Over the course of the years since the original DQN paper was published, there have been numerous proposals aiming to improve DQN. We discuss the most influential of them in this subchapter.

- **Prioritized experience replay** is an improvement over uniform batch selection from the experience replay buffer proposed in [60]. The key idea here is to select more "controversial" (with more difference between target prediction and actual values) samples more often, thus speeding up the convergence.
- **Double DQN** is a technique that helps reduce the overestimation bias of the Q-network proposed in [61]. Since in basic DQN the same network is used for both action selection and action evaluation, the model is prone to overoptimistic value

estimates. By decoupling these tasks into two separate networks, this improvement results in faster convergence and better performance in specific RL problems.

- **Multi-step learning** is an extension of the basic policy bootstrapping of the original Q-learning algorithm. In the base Q-learning (and subsequently DQN) the estimation update function only considers single step lookahead. By looking more steps ahead one can improve agent performance in cases where precise sequences of action matter and achieve an overall faster learning.

Distributional Deep RL

Distributional Deep RL is one of the more recent advancements in the field of Deep RL [62]. The key idea of distributional RL is approximating the full distribution of returns rather than its expectation. Bellemare et al. introduce a random variable $Z(s, a)$ which represents the reward aquired by starting from state s, performing action a and following current policy from there. Action-value function can be redefined in terms of $Z(s, a)$:

$$Q^{\pi}(s, a) = \mathbb{E}[Z^{\pi}(s, a)] \tag{2.33}$$

Using variable $Z(s, a)$, the Bellman equation can be rewritten in distributional form:

$$Z^{\pi}(s, a) \stackrel{D}{=} R(s, a) + \gamma Z^{\pi}(s', a^*) \tag{2.34}$$

Here $s' \sim p(\cdot|s, a), a^* = arg\max_{a'} \mathbb{E}[Z^{\pi}(s', a')]$. Using this formula of *value distribution* authors proposed to work on optimization of distributions, reducing loss between current value distribution function approximation and target value distribution. As metric for measuring loss, the *Wasserstein metric* is proposed.

Based on the original C51 paper [62], Dabney et al. proposed Implicit Quantile RL algorithm (IQN) [63]. This algorithm also operates with value distributions, but unlike C51, its network outputs single sample from this distribution instead of fixed atomic ranges. The algorithm transforms the state into a vector V with fixed dimensions, then uses a randomly sampled scalar τ to produce a vector H of the same shape as V using the function $\phi(\tau)$:

$$\phi_j(\tau) = ReLU(\sum_{i=0}^{n-1} cos(\pi i \tau) w_{ij} + b_j) \tag{2.35}$$

This vector is then combined with V by concatenation or multiplication (depending on the shape of the remaining layers) and then fed forward to the rest of the deep layers, producing a vector with the size $|A|$ which represents a single sample from the distribution for each action. To get better fidelity, forward passes of IQN are re-run multiple times to get more samples of distribution.

Distributional RL provides multiple state-of-the-art algorithms, their short overview is presented in Figure 2.21

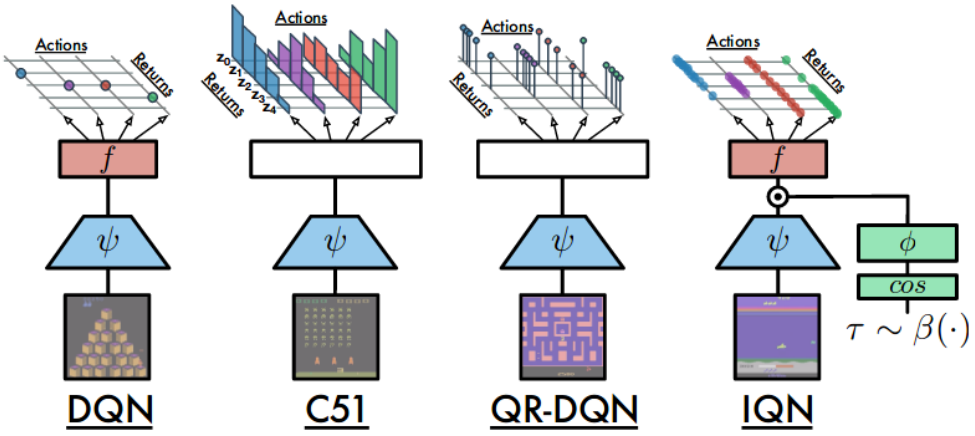


Figure 2.21: Network architectures for DQN and recent distributional RL algorithms [63]

In our work we use the Rainbow DQN agent, which consists of multiple improvements over the regular DQN algorithm described, with a Distributed DQN approach based on KL loss, and an IQN agent. Both of them are based on implementations provided as part of Google Dopamine framework.

2.4 Summary

In this chapter we provided an overview of the domain of our research. First, we described the core concepts behind physical database design and database fragmentation in particular. Then we focused on the task of finding optimal horizontal fragmentation and classical as well as state-of-the-art algorithms for solving this problem. We finish with an overview of machine learning methods, with detailed description of clustering and reinforcement learning tasks.

3. Prototype implementation and research questions

This chapter is devoted to the design of our proposed ML solutions for horizontal partitioning, and the adaptation of classical algorithms. This chapter is structured as follows:

- In Section 3.1 we start by formulating the questions we intend to answer with our research.
- In Section 3.2 we proceed by outlining the framework for training and evaluation of our solutions.
- In Section 3.3 we describe the design of the cost model, used by our solutions and in the classical algorithm.
- Section 3.4 provides an outline of how we adapted the classical algorithm for horizontal fragmentation to fit the evaluation requirements.
- In Section 3.5 we describe the design of the proposed Clustering-based ML solution for horizontal fragmentation.
- In Section 3.6 we describe the design of the proposed Deep-RL-based ML solution for horizontal fragmentation.
- We conclude the chapter by summarizing its contents.

3.1 Research questions

In Section 1.2 we define the general research structure for this thesis. In this chapter, implementation-specific questions are included in each solution's sub-chapter.

3.2 General structure

In this chapter we introduce and explain the general structure of our horizontal fragmentation software system. The system can be divided into stand-alone parts as shown in Figure 3.1.

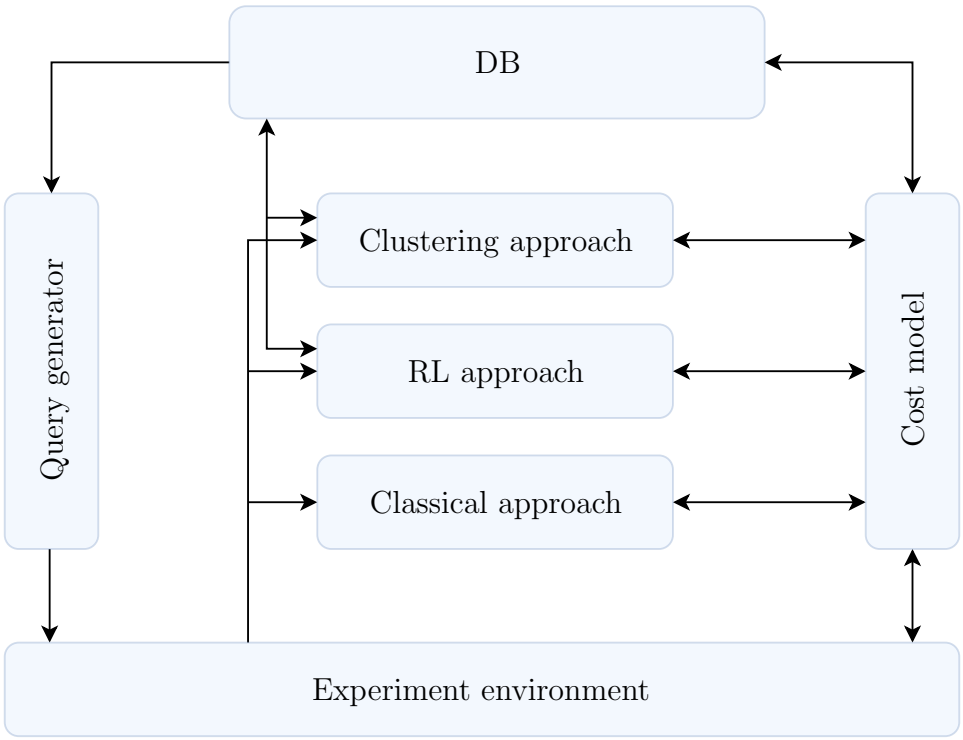


Figure 3.1: General structure of the software system

We implement and compare three approaches for horizontal fragmentation: clustering-based, RL-based and an approach based on one of the classical algorithms for horizontal fragmentation described in Section 2.2. All of the evaluated approaches use a cost model to guide the fragmentation process. The cost model estimates the execution cost of the queries based on the fragmentation scheme generated by the approaches. In addition, the clustering-based and RL-based algorithms use the database directly, as will be explained in the next sections. The query generator generates a set of queries for the experimental environment taking into account the distribution of the attribute values in the database. Implementation details of the components mentioned above will be discussed in the next sections.

3.3 Cost model selection

Both ML approaches for horizontal fragmentation proposed in this thesis as well as the classical algorithms need to create and tear down fragmentation repeatedly to optimize them. Originally we intended to use PostgreSQL and its checks-based fragmentation

semantics. To set up the partitions one needs to define a "master" table, which will serve as a schema description for all fragments. This table will hold no data. Then fragment tables can be created, inheriting from the "master" table. Fragment predicates should be stated in CHECK statement of the child table. Finally insert triggers need to be added to make sure that all data added to the master table gets redirected to fragments (select, update and delete queries get redirected automatically due to the nature of inheritance relation). The fragment creation operation has the following syntax:

```
CREATE TABLE test(...);
CREATE TABLE test_fragment_1(CHECK (predicate_1)) INHERITS (test);
CREATE TABLE test_fragment_2(CHECK (NOT predicate_1)) INHERITS (test);
```

However, in the course of developing this thesis we have encountered a major challenge: it turned out that continuous recreation of fragments of the same table leaves over temporary data from previous fragmentations. There is no effective way to manage this redundant data from PostgreSQL and continuous vacuuming after each test introduced too much overhead to keep working on algorithms.

Because of that we have decided to design a cost model that would work closely with the original database, but would avoid the creation of actual data structures that PostgresSQL needs to keep track of the fragments and without copying data to child tables.

The main concern of design of our proposed cost model is imitating the query optimizer of a modern DBMS while highlighting the reduction in CPU work and I/O caused by fragmentation. In fragmented database query planner can make use of the meta-data from fragment tables, allowing to skip loading some of them, thus skipping data fetching and processing costs altogether [64].

The process of this optimization is applied as follows. Incoming queries can evaluate their filter predicates against the predicates of fragments and find the correlations, which decide if the data inside the fragment needs to be loaded or not. For example, if the query has the filter predicate `age = 25`, and the fragment predicates contain predicate `age < 20`, fragment can be safely marked as skippable.

We assume that the DBMS has unlimited time budget for determining such correlations between query and fragment predicates and can find implications in complex query filter conditions. Algorithm for determining if the fragment can be skipped for general case can be seen in Algorithm 6:

Algorithm 6: Determining if fragment is eligible for skipping

Input: fragment predicates F, query predicates Q, query logic tree T_{query}, fragment logic tree $T_{fragment}$
Initialise independent predicates list $V \leftarrow \emptyset$
Initialise fragment predicate map $M\{f \mapsto \textbf{bool}\} \leftarrow \emptyset$
Set solutions $S \leftarrow SAT(T)$
for each q **in** Q **do**
  if q in all $s \in S$ **and** value of q is same across all $s \in S$ **then**
    add q to V
  end if
end for
for each v **in** V **do**
  for each f **in** F **do**
    if $v \implies$ **not** f **then**
      add $\{f \mapsto \textbf{false}\}$ to M
    else if $v \implies f$ **then**
      add $\{f \mapsto \textbf{true}\}$ to M
    end if
  end for
end for
Substitute the support variables in $T_{fragment}$ with M
return $T_{fragment}$ simplifies to **false**

Both query and fragment are represented by two major parameters: set of atomic column predicates, encoded as `<column> <operand> <rvalue>` and tree of logic tokens **and**, **or**, **not**, which represent the relations between the predicates in filter statement. This algorithm allows to find the correlation between two differently combined sets of predicates by determining the set of independent (required for the query filter condition to be true) values of query predicates, finding the implications between these values and atomic fragment predicates, and computing the actual value of fragment predicate given these implications.

The cost model designed as part of this thesis can operate on all atomic predicates with one of the following operations: $=, \neq, <, \leq, >, \geq$. This corresponds to the query workloads used for training and evaluation of algorithms as discussed in Chapter 4.

We subscribe to the assumption that the cost of query execution is proportional to the amount of data that needs to be fetched from disk, as the amount of CPU processing is negligible. This amount equals to `tuples_fetched * size_of_tuple`. However, for measuring the performance of the queries on the same table `size_of_tuple` can be dropped.

3.4 Classical algorithm adaptation

Using clustering or RL for horizontal fragmentation is not an end in itself, so when analysing these approaches they should not be compared against each other, but rather one should compare them against already existing "classical" algorithms for horizontal fragmentation. That is why some of the articles that were discussed in Section 2.3.2.2 use a classical horizontal fragmentation algorithm to evaluate the proposed approach.

However, to use a classical fragmentation algorithm as a fair baseline, the algorithm needs to be adapted to the limitations and specific features of the task domain. For instance, comparing the results of the algorithm proposed by Cuzzocrea et al. [46] with the results obtained from predicate-based and affinity-based algorithms is not fair, because the classical algorithms without modifications can not limit the number of fragments.

So in order to choose a classical horizontal fragmentation algorithm for a baseline we need to specify the requirements, which the algorithm needs to satisfy:

- it should be possible to put a constraint on the number of fragments generated by the algorithm, because it is a limitation of our RL-based approach;
- decisions made by the algorithm should only be based on the queries and their access to the data;
- the algorithm should be applicable for a non-distributed environment;
- ideally, the algorithm should use a cost function to guide the fragmentation process, at least to some extent.

From these and other considerations (e.g. complexity of the algorithms) the algorithm proposed by Zhang and Orlowska [1] was selected as a baseline. This approach can be perfectly adapted to the requirements given above. We have configured the algorithm as follows:

- to obtain k fragments, $k-1$ points along the main diagonal of the predicate affinity matrix are selected by trying all possible $k-1$ combinations of the points;
- the optimal combination is selected using a cost function; hence, in this way we incorporate the cost function into the decision process;
- although using the `SHIFT` procedure increases the algorithm complexity, we use it to generate the best results possible with this algorithm;
- as mentioned in the Chapter 1, we assume that all the queries are executed equally often, that is why the frequency of each transaction is set to 1.

3.5 Clustering-based solution

3.5.1 Specific research questions

Similarly to the classical fragmentation approach, there are requirements that our clustering-based fragmentation approach must fulfill. However, there are more options to choose from and all the decisions made (e.g. on input data representation, clustering algorithm, data structures used, etc) must comply with the requirements:

1. Our clustering-based approach should be comparable to the RL-based approach. That means that we need to take into account limitations and characteristics of the RL-based approach:
   - the number of generated fragments should be fixed;
   - the RL-based approach uses a cost model to take an action, therefore the clustering-based approach should also be able to use the cost model to make decisions.
2. Besides that, it should also be:
   - fast;
   - transparent;
   - case-specific.

Based on these requirements we formulate our implementation-specific research questions for the clustering-based approach as follows:

1. Which of the clustering algorithms discussed in Section 2.3.2 is the best fit for our task considering the requirements stated above? How and to what extent can the complexity of the selected clustering algorithm be reduced?
2. Which input data representation can speed up the execution of the selected clustering algorithm and guarantee completeness, reconstructability and disjointness of the obtained fragmentation scheme?

3.5.2 Clustering algorithm selection

The selection of a clustering algorithm for the task is an essential decision in our case. It must be efficient, but simple and transparent at the same time. All the types of clustering algorithms presented in Section 2.3.2 have their drawbacks and benefits in the context of our task:

- partitioning-based algorithms:

  + fast;
  + easy to implement;
  + the number of fragments can be limited;
  − it is hard to include a cost model into the decision process, because partitioning-based algorithms make a decision at the cluster-point level, whereas a cost model is used to compare the quality of two complete fragmentation schemes;
  − the clustering process is not deterministic and the results depend on the selection of initial centers.

- hierarchical-based algorithms:
  + a cost model can easily be included into the process of clustering;
  + the number of fragments can be limited;
  + these algorithms do not have parameters that need to be chosen based on deep domain expertise;
  + they are easy to implement and the mechanism of the clustering process is transparent and deterministic;
  + such algorithms are flexible in terms of a variety of similarity measures and linkage criteria, that can be used;
  − hierarchical-based algorithms have high computational complexity.

- density-based algorithms:
  + robust to outliers;
  − it is not clear, how to select suitable values for parameters, which characterize fragments as the areas of high density (e.g. ε and $minPts$ for DBSCAN algorithm) and it does not conform to the transparency requirement;
  − it is hard to put a constraint on the number of fragments;
  − it is hard to include a cost model into the clustering process.

- grid-based algorithms:
  + perform quite well on large datasets;
  + a cost model can be included into the fragmentation process;
  − finding a suitable size of the grid is a difficult task;
  − these algorithms are not very accurate [65], which can drastically degrade the quality of results.

- model-based algorithms:
  + might generate better results than partitioning-based clustering algorithms [29];

+ the number of fragments can be limited;
− to choose suitable values for the parameters of model-based clustering algorithms, the analyst should have certain knowledge of the underlying data distribution, which is hard in our case;
− they have high computational complexity.

Taking into consideration these characteristics of the clustering algorithms, we have chosen a hierarchical-based agglomerative clustering algorithm for our clustering-based horizontal fragmentation approach. The algorithm is transparent, flexible, automatic in the sense that it does not require a thorough analysis of the dataset. Furthermore, high computational complexity of the algorithm can be reduced by using a heap, where distances between cluster pairs are stored.

The heap is an implementation of a priority queue, which requires an element with higher priority to be served before the one with a lower priority. So the element with the highest priority is always placed at the root of the heap and the process of finding this element has $O(1)$ complexity. Deletion of an element from the heap and insertion an element to the heap usually have $O(log\ n)$ complexity. So the time complexity of the heap-based agglomerative hierarchical clustering algorithm is $O(n^2\ log\ n)$. With all the improvements and adjustments our hierarchical clustering algorithm works as follows:

1. **Input data preparation.**

   For details, please refer to the Section 3.5.3.

2. **Generation of the initial clusters.**

   Each data point from the step 1 is put to its own cluster.

3. **Building a heap.**

   Using the specified linkage criterion and similarity measure, all the distances between each pair of the clusters are calculated and added to the heap.

4. **Getting cluster pairs from the heap.**

   The algorithm repeatedly gets a pair from the heap and if both clusters of the pair are valid (some clusters in the heap can be already merged before) adds it to a list of cluster pairs under consideration.

5. **Best pair selection.**

   The algorithm tries to merge each pair of clusters from the list of the cluster pairs under consideration and using a cost model compares execution cost of the queries on the modified fragmentation schemes.

6. **Updating the clusters.**

   The best pair of clusters from the step 5 is merged, other pairs are pushed back to the heap.

7. **Repeat steps 4-6** if the current number of clusters is less than the specified number of clusters k.

3.5.3 Input data representation

Although the heap-based agglomerative hierarchical clustering algorithm is much faster than the classical version, substantial increase in speed of the algorithm can only be achieved by using an optimized input data representation.

From the problem statement (rows of the database table should be grouped based on their usage in the queries) an obvious, straightforward solution would be to encode each row using its access statistics. But are there better options? To choose the best one, different input data representations from the papers discussed in Section 2.3.2.2 need to be analyzed.

Adrian Darabant and his scientific group use object-condition and characteristic vector matrices as their input data representations. If $Pred(C)$ is a set of atomic predicates extracted from the queries and $Inst(C)$ is a set of instances of the class C, then the object-condition matrix can be constructed as follows:

$$OCM(C) = a_{ij}, 1 \leq i \leq |Inst(C)|, 1 \leq j \leq |Pred(C)|, \tag{3.1}$$

where a_{ij}:

$$a_{ij} = \begin{cases} 1, \text{if predicate } j \text{ accesses instance } i \\ 0, \text{otherwise} \end{cases} \tag{3.2}$$

Each element in the characteristic vector matrix represents the ratio between the instances of the class C, which are accessed in the same way as instance i by a predicate j is, and the total number of instances of the class C.

For the purpose of the discussion we will call such input data representations object-predicate representations after the level of its granularity. One advantage of such representations is that the fragmentation algorithm proposed by Zhang and Orlowska [1] is also predicate-based in a sense, so the algorithm proposed by Zhang and Orlowska would be more comparable to our clustering approach if this input data representation was used. The same input data representation, but for tuples instead of objects, was used in other works [50, 51]. However, such representations only make sense if there are predicates that are used in several queries. Moreover, they do not handle well the situation when two different predicates fetch the same or similar sets of rows (objects), as they only compare predicates syntactically. It is also not always an easy task to extract and to analyze predicates from the queries. Object-/tuple-based component of the representation is not a perfect solution either. For a table with 1 million objects/tuples running an algorithm with $O(n^2\ log\ n)$ complexity on the input data represented this way would take too much time.

Other input data representation called predicate usage matrix was used in several works [46–48, 53]. The matrix has queries as rows and predicates extracted from the queries as columns. A cell (i, j) value is set to 1 if a query i includes a predicate j and 0 otherwise. Representing the data in this way makes the algorithm work even more similar to the algorithm proposed by Zhang and Orlowska [1]. This approach has the same problems as the object-predicate representation does. And not to mention that by using this input data representation, the reconstructability and disjointness of the obtained fragments can not be guaranteed automatically (without using some post-processing). Both data representations have scalability problems: even if the number of queries is small, the queries might consist of a huge number of simple predicates and if there are not so many queries sharing the same simple predicates, the matrix will have an enormous number of columns and still will not provide enough data for clustering to be meaningful Table 3.1.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
q_1	0	0	0	0	0	1	1	0	0	0
q_2	0	0	0	1	1	0	0	0	0	0
q_3	1	0	1	0	0	0	0	0	0	0
q_4	0	1	0	0	0	0	0	1	1	1

Table 3.1: Predicate usage matrix

So by analyzing different input data representations from the literature we define our requirements as follows:

- our input data representation should guarantee correctness properties of a horizontal fragmentation scheme generated by the hierarchical clustering algorithm automatically;
- our input data representation should not be based on the simple predicates from the queries because of the reasons mentioned before;
- our input data representation should be faster than the tuple-/object-based input representations, since our testing table is quite large.

All these requirements can be satisfied very easily by analyzing the problem again: tuples, which are frequently accessed together in the queries, should be moved to the same fragment. That means, that it does not make sense to consider each row separately. Instead, we can use as an input data representation for the clustering algorithm sets of tuples, which behave in the same way on each single query from the list of queries. We

will call such sets atomic clusters (or atomic fragments), since they are on the lowest level of granularity that is reasonable to use in the context of our task. For n queries there are 2^n possible combinations of the queries that need to be considered. To generate a set of atomic fragments, a `SELECT` query needs to be constructed from each combination of the queries: e.g. taking combination [0, 0, 1] of set Q of queries $\{q_1, q_2, q_3\}$ following query will be formed:

$$\neg q_1 \wedge \neg q_2 \wedge q_3 =$$
$$\texttt{SELECT * FROM <table> WHERE NOT <}q_1\texttt{> AND NOT <}q_2\texttt{> AND <}q_3\texttt{>;} \quad (3.3)$$

The rows selected by this query will form an atomic fragment and [0, 0, 1], which will become its fragment condition vector. A row can either be selected or not selected by a query and the queries in normal or negated forms are connected by the `AND` operator, which means that the predicates generated from the fragment condition vectors are mutually exclusive and one row can be assigned only to a single atomic fragment. By considering all the possible combinations of the queries we also guarantee that each row will be assigned to at least one atomic fragment.

So as input data representation we use a set of atomic fragments (Table 3.2). Each atomic fragment also has information about the number of rows assigned to it.

amount of rows selected	queries		
	q_1	q_2	q_3
2006	0	0	0
0	0	0	1
1005	0	1	0
366	0	1	1
0	1	0	0
5001	1	0	1
99	1	1	0
800	1	1	1

Table 3.2: Atomic fragments

For instance, the Table 3.2 shows that there are 2006 tuples in the database table which are selected by the predicate of the first atomic fragment($\neg q_1$ & $\neg q_2$ & $\neg q_3$). The data is prepared for the clustering algorithm as follows:

1. atomic clusters with 0 rows assigned to them are eliminated;
2. to each non-empty atomic cluster a unique id is assigned (Table 3.3).

amount of rows selected	queries			id
	q_1	q_2	q_3	
2006	0	0	0	0
~~0~~	~~0~~	~~0~~	~~1~~	-
1005	0	1	0	1
366	0	1	1	2
~~0~~	~~1~~	~~0~~	~~0~~	-
5001	1	0	1	3
99	1	1	0	4
800	1	1	1	5

Table 3.3: Atomic fragments after processing

Input data can also be presented in Euclidean space by considering the fragment condition vector of a cluster as its coordinate (Figure 3.2).

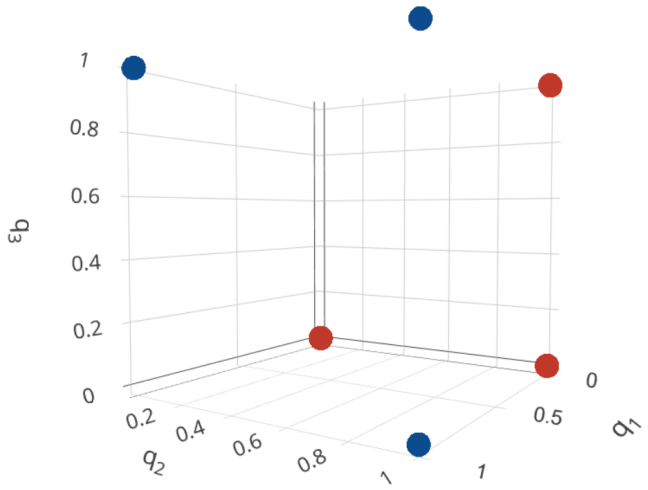


Figure 3.2: Input data in Euclidean space

In agglomerative hierarchical clustering, the initial data points are moved to the separate clusters at the beginning. So our clustering algorithm uses the set of atomic fragments as initial clusters. In the next steps of the algorithm if two clusters are merged, the rows stored previously in two different clusters (fragments) are now stored together. This implies several things:

- id of a new cluster must contain information about the atomic clusters, which were assigned to the old two clusters;
- the number of rows in the new cluster is equal to the total amount of rows assigned to the old clusters;
- predicates generated from fragment condition vectors of the old clusters should be `OR`ed to obtain a predicate of the new cluster, since in the new cluster rows which are selected by the predicate of the first cluster `OR` by the predicate of the second cluster are stored.

Internally we represent the predicate of the new cluster as an array of fragment condition vectors of the atomic clusters belonging to it. If a cluster c is encoded like $\{id, predicate, rows\_amount\}$ and two clusters $c_1 = \{0, [0,0,0], 2006\}$ and $c2 = \{1, [0,1,0], 1005\}$ are merged, a newly generated cluster c_3 will be represented as follows: $(\{\{0,1\}, [[0,0,0], [0,1,0]], 3011\})$.

When the required number of clusters is reached and fragments are generated from these clusters, array of fragment condition vectors like [[0, 0, 0], [0, 1, 0]] can be optimized following Boolean rules for simplification and obtained fragment predicate will look like:

$$\neg q_1 \wedge \neg q_2 \wedge \neg q_3 \ \vee \ \neg q_1 \wedge q_2 \wedge \neg q_3 = \neg q_1 \wedge \neg q_3 \tag{3.4}$$

Analysis: initially all the combinations of the queries should be considered, which limits the size of input dataset (the number of atomic clusters) to 2^q, if q is the number of queries. Theoretically, the selected input data representation can only guarantee that this number will always be less or equal to the number of rows in the real database table, but in reality the number of atomic clusters will be much smaller. It does only make sense to select up to 10 queries with the "80/20 rule" mentioned in Section 2.2, since neither of the approaches used in this thesis (the classical fragmentation algorithm, the clustering-based approach and the RL-based approach) works well on a very large set of queries or set of the simple predicates.

By using this input data representation we make our clustering algorithm run much faster, than it would if tuple-based data representation was used. Moreover, by considering all relevant combinations of the queries we guarantee completeness and disjointness of the generated fragmentation scheme, which could not be achieved by using any other input data representation from the literature.

3.5.4 Similarity measures and linkage criteria

Darabant et al. [41] present their research on the influence of different similarity measures on the results obtained by clustering-based fragmentation. They use only 3 similarity measures: Cosine, Euclidean and Manhattan. The results show, however, that the impact of the similarity measures depends on the problem statement itself (primary or derived horizontal fragmentation), the methods of constructing vectors, etc. We also assume that the results could be completely different, if another cost model or linkage criterion was used. One of the reasons for that lies in the inability of hierarchical clustering algorithms to undo actions taken before, which increases the difference between the results generated using different similarity measures at each iteration. Therefore, we believe that choosing the right combination of similarity measure and linkage criterion for the selected cost model and input data representation is an incredibly important task for us. Generally, this also means that there is no optimal combination of similarity measure and linkage criterion for all of the cases. Considering that hierarchical clustering allows pretty much all existing similarity measures and linkage criteria to be used, we would like to test as many combinations of them, as we can within a limited time frame. In order to do that we support the following linkage criteria:

- Single-linkage criterion;
- Complete-linkage criterion;
- Average-linkage criterion;
- Centroid-linkage criterion.

And the following similarity measures:

- Euclidean distance;
- Squared Euclidean distance;
- Manhattan distance;
- Maximum distance.

Furthermore, for the hierarchical clustering algorithm application-specific linkage criteria and similarity measures can be defined, which allows us to create case-specific, well-tailored to the horizontal fragmentation problem and selected input data representation linkage criteria and similarity measures. We have developed our own method of measuring the distance between two clusters, called *Penalty-based method*.

This method is based on the intuition behind horizontal fragmentation. Consider a fragment, which consists of multiple atomic fragments and set $Q = \{q_1, q_2, \ldots q_n\}$ of

queries. The fragmentation predicate provides an information to DBMS on whether or not the fragment is selected by a query. All the rows from the fragment are fetched by a query q_i if at least one its atomic clusters has 1 in the ith element of its fragment condition vector. If all of the atomic clusters are placed in their own clusters/fragments, the total execution time of the queries Q is minimal. When two fragments are merged, execution time of the queries increases. The reason is that after merge there are going to be atomic clusters in the newly merged cluster, that are accessed differently by some queries, i.e. some queries will fetch all the rows from the merged fragment even though some of them are not relevant. The number of rows that were fetched unnecessarily is the penalty for merging two fragments.

Using penalty-based distance measure the distance between two fragments is calculated as follows:

1. Atomic clusters from the two fragments are merged together into a fragment F.
2. For all of the queries, which fetch some of the rows in F, we analyze the atomic clusters one by one and sum the number of rows from the atomic clusters, which have 0 in the corresponding element of the fragment condition vector (Table 3.4).

amount of rows selected	queries			
	q_1	q_2	q_3	q_4
2006	0	0	0	0
366	0	1	0	0
5001	1	0	0	1
Total penalty = 2006 * 3 + 366 * 2 + 5001 = 11751				

Table 3.4: Penalty-based method

3.6 Deep-RL-based solution

3.6.1 Research questions

From the assumptions listed in Section 1.1 and general questions from Section 1.2 we derive a set of implementation-specific research questions the design needs to answer:

- How to formulate the task of finding the right horizontal fragmentation in terms of RL? This includes designing the environment, observation and action spaces.
- How does the logic of the environment traversal impact the convergence of the model?

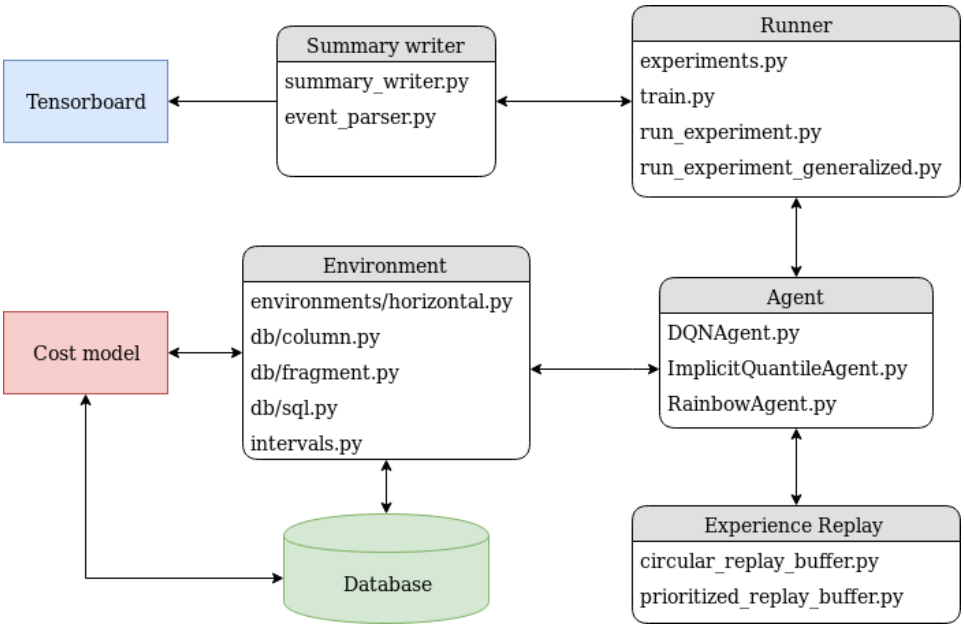


Figure 3.3: Architecture of Deep-RL based solution

3.6.2 Architecture

The implementation of our Deep-RL solution is based on Dopamine framework developed by Google. Dopamine is designed to ease the prototyping and evaluation of Deep-RL algorithms, and includes a number of state-of-the-art algorithms (including DQN, Rainbow, Implicit quantile) [66]. This framework uses environment interface from Gym toolkit, which provides a set of standardized environments as well as means to define new environments [67].

The general architecture of the Deep-RL solution, based on original Dopamine framework architecture, is shown in Figure 3.3

1. **Runner** is a key component that defines the routines for single training or training-and-evaluation task. It orchestrates the training and evaluation process by connecting agent and environment, accumulates and processes the statistics for logging. Here we also include scripts for starting and managing multiple runners in series of experiments.

2. **Environment** is designed to encapsulate all the domain-specific business logic of horizontal fragmentation, providing standard Gym interface with extensions for action pruning. It serves as a bridge between ML model and database, translating the changes to target fragmentation to concepts of RL task (observation, action).

3. **Cost model** is designed to improve experiment run time as discussed in Section 3.3. It uses the target database and assumptions on query planner's optimizations to calculate costs of executing query workload on a given fragmentation.

4. **Agent** is a component that represents the RL entity of the same name. Dopamine framework comes with a number of easily adaptable agents, from which we use Rainbow and Implicit Quantile. The agent behaviour was extended to accommodate action pruning.

5. **Experience replay** is an optimization technique commonly used in Deep-RL algorithms of the DQN family. High performance implementations of replay buffer are provided in Dopamine framework. Some minor changes were done by us for it to fit the required data format.

6. **Summary writer** is a component, which is tasked with filtering and storing experiment results. We use Tensorboard to visualize and group data, and this component preprocesses the data for it.

3.6.3 Input data representation

In order for RL-algorithms to be applicable, the optimal horizontal fragmentation task needs to be re-framed in RL terms. This means defining the implementation details of the following concepts:

- **Observation**: the information about current state of the task the RL agent solves. In our specific case, it should include the information about the database, queries and current fragmentation.
- **Action**: the information about how to transition between states. This part is especially hard because the choice of action representation also limits the possible fragmentation the agent can create by taking them.
- **Reward**: the information about how good the action taken was.

To start designing the observation space for the horizontal fragmentation we need to consider the limitations of the Deep-RL agents. The observation space of all Deep-RL agents used in this thesis represents a multidimensional finite-sized field, where values are typically of the same magnitude.

The observation should encode the information about the data in table, so that the agent can use it to find the patterns to be used in fragmentation. However, the typical size of a database is huge by the standards of the RL agents (most of the agents used in this thesis are designed to interact with the environment with 160 by 210 matrix of 7 bit pixels as state observation and 18 discrete actions [68]). The database can also change its size when the user adds or deletes tuples. This makes it unwieldy for RL agent to operate on the tuple level.

However, the agent does not need the *full* information on the data in table. It is mostly interested in how the data interacts with queries and fragments. Authors in [69],

when faced with similar problem for the index selection task, proposed the following representation of the query workload with regards to the selectivity of each column used in the query:

$$I_{workload} = \begin{bmatrix} Sel(q_1, c_1) & \dots & Sel(q_1, c_n) \\ \vdots & \ddots & \vdots \\ Sel(q_k, c_1) & \dots & Sel(q_k, c_n) \end{bmatrix} \tag{3.5}$$

for queries $q_i \in Q$, columns $c_j \in C$ and:

$$Sel(q, c) = \begin{cases} \frac{\text{number of tuples selected by } q \text{ on } c}{\text{total number of tuples in the table}}; & \text{if } q \text{ has predicates on } c \\ 1; & \text{if } q \text{ no has predicates on } c \end{cases} \tag{3.6}$$

This, however, requires multi-staged preparation of query workload to separate predicates column-wise and calculate their selectivity separately. Such approach also hinders the ability of the agent to find correlation between the queries and their predicates.

Assuming that each predicate of query is a range selection on specific column, we can instead provide the agent with the raw information about predicate itself. We can define the query workload input data representation with same shape to [69], but with predicate ranges:

$$I_{workload} = \begin{bmatrix} [R_{left}(q_1, c_1); R_{right}(q_1, c_1)] & \dots & [R_{left}(q_1, c_n); R_{right}(q_1, c_n)] \\ \vdots & \ddots & \vdots \\ [R_{left}(q_k, c_1); R_{right}(q_k, c_1)] & \dots & [R_{left}(q_k, c_n); R_{right}(q_k, c_n)] \end{bmatrix} \tag{3.7}$$

where

$$\begin{aligned} R(q, c) &= [R_{left}(q, c); R_{right}(q, c)] \\ &= \begin{cases} [\text{left limit point}; \text{right limit point}] & \text{if } q \text{ has predicates on } c \\ [\texttt{COLUMN\_MIN}, \texttt{COLUMN\_MAX}]; & \text{if } q \text{ no has predicates on } c \end{cases} \end{aligned} \tag{3.8}$$

Defining the observation in such way helps limit the dimensions of the problem by condensing them to the size of $|Q| * |C| * 2$. The same principle can be applied to

representing the fragments. Taking into account the limitation of number of fragments, we can encode the fragment predicates in 3-dimensional table of size $F_{max} * |C| * 2$ where F_{max} is the maximum number of fragments.

The Deep-RL solutions usually operate on uniform data, however, data in columns (and, consequently, in column ranges) depends on the column type. To make the input data uniform, we propose to normalize it using the following formula:

$$x_{normalized} = \frac{x - \texttt{COLUMN\_MIN}}{\texttt{COLUMN\_MAX} - \texttt{COLUMN\_MIN}} \tag{3.9}$$

This way, any value x can be brought to range of $[0; 1]$, which can then be easily consumed by Deep-RL agents. We additionally specify single column per query/fragment which shows the selectivity of said query/fragment. The final shape of observation is $(|Q| + F_{max}) * (|C| * 2 + 1)$.

However, this way of encoding ranges is very restrictive in number of values it can represent. For encoding the fragments one key type of range is an inverse interval, which is useful when defining negation of predicates for co-fragments to guarantee the disjointness and completeness requirements. In current encoding, a negation of a predicate range $[a, b]$ will, in general case, result in two fragments: $[0, a)$ and $(b, 1]$. This hinders the ability of the agent to create the co-fragments while being restricted on maximum number of fragments. To work around this, we introduce *inverse ranges* which are encoded as $[b, a]$ where $b > a$. Since regular ranges with left limit point being higher than right limit point do not make much sense (these two limits contradict each other and, therefore, yield when used in filter statement of query or fragment), we can safely use encode the negated predicates this way and rely RL agents to recognise the pattern.

We also need to highlight the empty fragments for the agent. Fortunately, there is still ranges of $[0, 0]$ unused and not encoding any meaningful intervals. An example of single observation encoded in such way can be seen on Table 3.5:

c_1		c_2		c_3		c_4		sel	
0	1	0.245	0.65	0	1	0	0.4	0.1	queries
0.33	0.98	0	1	0	1	0	1	0.4	queries
0.1	0.25	0	1	0	1	0	1	0.3	fragments
0.25	0.1	0	1	0	1	0	1	0.7	fragments
0	0	0	0	0	0	0	0	0	empty fragments
0	0	0	0	0	0	0	0	0	empty fragments

Table 3.5: Example of observation space for 2 queries and 4 maximum fragments

3.6.4 Action representation

Given that the RL agent is tasked with finding the optimal horizontal fragmentation, it needs to be able to create and update fragments. This maps to **actions** concept of RL: the atomic changes in observed state chosen by agent, which give some reward. From the Section 3.6.3 it is clear that actions should change defined ranges, updating them or creating new ranges if the fragment is already partitioned.

However, actions represented by arbitrary changes to continuous values create huge action space, which proves tricky for Deep-RL agents to learn. We propose using atomic filter predicates of the queries from query workload to form action. This way the learning process starts from single proto-fragment (representing unfragmented table) and then agent selects predicate-fragment pairs to form new fragments or update existing.

The mechanism of applying the predicate $[a, b]$ on column c to fragment with current number of allocated fragments $F_{current}$ has following steps:

1. Both $[a, b]$ and the fragment predicate $[f_{left}, f_{right}]$ on c are converted to mathematical interval forms i_{query} and $i_{fragment}$ (inverting the predicate if the left limit point is higher than right).

2. $i_{positive}$ is calculated as $i_{positive} = i_{query} \cap i_{fragment}$. If $i_{positive}$ is the same $i_{fragment}$, the process of fragment splitting is ended prematurely, signaling the agent that this action does not produce meaningful result.

3. $i_{negative}$ is calculated as $i_{negative} = \neg i_{query} \cap i_{fragment}$. If the $i_{negative}$ is empty interval, the process of fragment splitting is ended prematurely, signaling the agent that this action does not produce meaningful result.

4. Both $i_{positive}$ and $i_{negative}$ are converted to state encoding. This might produce maximum of 3 different ranges using the inverted interval encoding discussed earlier. Lets mark the number of ranges created as F_{new}.

5. If $F_{max} < F_{current} + F_{new} - 1$, , the process of fragment splitting is ended prematurely, signaling the agent that this action does not fit the requirements on maximum number of fragments.

6. Otherwise, next $F_{new} - 1$ empty fragments are set to copy of the original fragments on all columns except c; it is instead set to its respective range from newly created ones. The original fragments has its c set to the first from set of new ranges.

Assuming that the queries have only a single range predicate on specific column, any action can be encoded as three values `query, column, fragment`. Action space of this approach is discrete and final, with maximum of $|Q| * |C| * F_{max}$ actions. An example of applying this kind of action is shown in Table 3.6:

c_1		c_2		sel
0.1	0.5	0.7	1.0	0.2
...				
...				
0.5	0.1	0.1	1.0	0.39
0	0	0	0	0

$\Longrightarrow$

c_1		c_2		sel
0.1	0.5	0.7	1.0	0.2
...				
...				
0.5	0.1	0.7	1.0	0.23
0.5	0.1	1.0	0.7	0.16

Table 3.6: Example of action based on query predicate

Here, action is highlighted in blue, presenting `query, column, fragment` triplet that defines the changes applied to fragmentation. Changes themselves are highlighted with green.

Reward engineering

To make a horizontal fragmentation optimizer based on RL, information about the quality of the defined fragments needs to be used as a reinforcement signal. Since the RL agent tries to maximize the *total* reward and we are interested only in the quality of the final fragmentation, it makes sense to withhold the reward until the terminal state, leaving it to policy bootstrapping to propagate the information about the fragmentation through states. The proposed method of rewarding the agent is presented in Figure 3.4

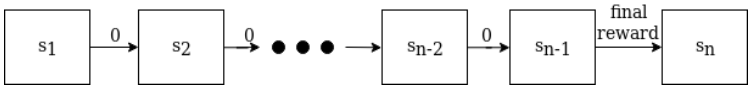


Figure 3.4: Schematic representation of rewards throughout the episode

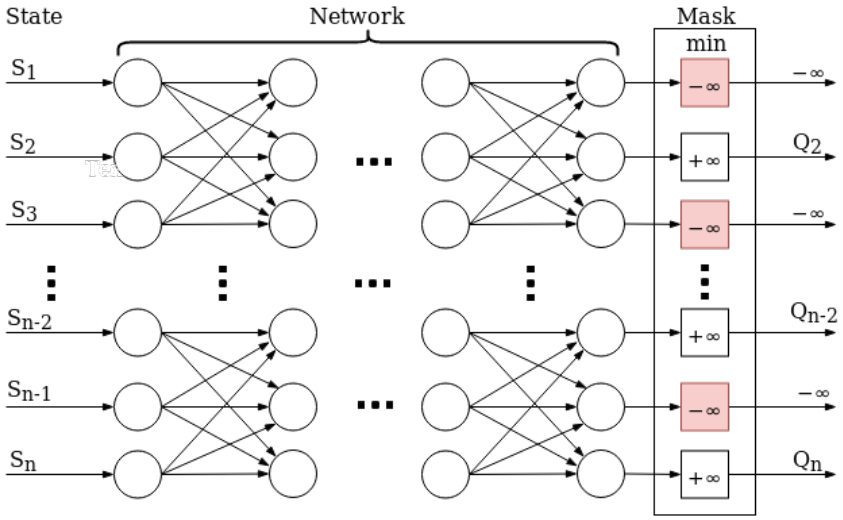


Figure 3.5: Schematic representation of action pruning embedded into Deep-RL agent

Since the task is to minimize cost of query execution, the reward function needs to decrease with the increase of cost. We propose the following function to be used to calculate the final reward in the end of the episode:

$$reward = max(\frac{cost_{unfragmented}}{cost_{fragmentation}} - 1; 0) \tag{3.10}$$

This function normalizes the performance for fragmentation with the performance on unfragmented database, cutting off all the results that perform worse then unfragmented database.

3.6.5 Action pruning

Taking some of the actions from the action space described in Section 3.6.4 results in either not changing state in any meaningful way, or changing the state into an invalid state. The traditional RL approach to such actions is to assign high negative rewards for taking them, teaching the agent to avoid them. However, another approach suggests embedding this knowledge into the architecture of the agent itself, thus improving the convergence [70]. To ensure that agent does not pick actions, which are invalid or useless, we propose the action pruning extension presented in Figure 3.5.

First step to employ action pruning is to build a mask at each step, which has the same shape as the action space. Each value of the mask is set to $+\infty$ if the action does not yield an invalid state, and the resulting state is different from current; otherwise value is

set to $-\infty$. When agent picks an action, the Q-values of each action are then replaced with $min(Q_i, mask_i)$, ensuring that actions that are invalid have zero probability to be picked during greedy step of the agent. We additionally filter actions during exploratory steps of the agent, picking random actions only from those, which have $+\infty$ in action mask.

3.7 Summary

In this section we outline the design decisions we made during work on our thesis. We specify the research questions we would like to answer with our work. Then we discussed the design on cost model used to evaluate the proposed solutions. After that, we described the designs of both clustering-based and Deep-RL-based solutions as well as the design decisions made to adapt the baseline classical algorithm.

4. Experimental design

In this chapter we discuss our experimental setup and provide some important details, which can be used for reproducing the experiments disclosed in our study. We organize the chapter as follows:

- In Section 4.1 we list important hardware and software characteristics of our experimental environment;
- In Section 4.2 we describe the dataset used for running the experiments;
- In Section 4.3 we discuss the query generator used to generate input data for the experiments and provide a sample set of queries generated by it;
- In Section 4.4 we list general settings of our horizontal fragmentation algorithms and some relevant configuration parameters of out RL-based solution in particular;
- We summarize the whole chapter in Section 4.5.

4.1 Experimental environment

We run our experiments with the classical horizontal fragmentation algorithm, clustering- and RL-based fragmentation algorithms on a device with the following characteristics:

- Memory: 32GiB
- Processor: Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz
- Graphics Card: GeForce GTX 1050 Ti Mobile
- OS: GNU/Linux Ubuntu 18.04.1

The list of software we have used and its versions:

- Python: 3.6
- PostgreSQL: 10.3
- psycopg2: 2.7.4
- pyeda: 0.28.0
- tensorflow: 1.14.0
- gym: 0.14.0
- Keras: 1.1.0
- dopamine-rl: 1.0.5
- scipy: 1.3.1
- gin-config: 0.2.0

4.2 Dataset

We use the `LINEITEM` table from the standard TPC-H benchmark. The TPC-H dataset is designed to have large data volumes and to represent a real-world business analysis industry case. The `LINEITEM` has the following attributes[1]:

- `l_orderkey` identifier;
- `l_partkey` identifier;
- `l_suppkey` identifier;
- `l_linenumber` integer;
- `l_quantity` decimal;
- `l_extendedprice` decimal;
- `l_discount` decimal;
- `l_tax` decimal;
- `l_returnflag` fixed text, size 1;
- `l_linestatus` fixed text, size 1;

[1] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

- `l_shipdate` date;
- `l_commitdate` date;
- `l_receiptdate` date;
- `l_shipinstruct` fixed text, size 25;
- `l_shipmode` fixed text, size 10;
- `l_comment` variable text size 44.

4.3 Workloads

Since in our thesis we aim to find an optimal way to fragment horizontally a database based on a set of queries and specifically based on a set of simple predicates (in case of classical and RL-based fragmentation algorithms) we would like to have a set of queries, which do not introduce features other than those, and which can be easily decomposed into a set of simple predicates. Therefore, queries of the TPC-H benchmark, which use a wide variety of complex operators and joins between the tables, are not really a perfect fit for us. That is why we have implemented a query generator, which generates a set of queries for the `LINEITEM` table and is designed according to such requirements:

- a query should be in the canonical form:

  `SELECT * FROM lineitem WHERE (<predicate> AND )* <predicate>;`

- RL-based approach requires a predicate to be in the form:

  `<column> '<' | '>=' <value>`

- we define predicates on numeric and date attributes of the LINEITEM table: `l_orderkey`, `l_partkey`, `l_suppkey`, `l_linenumber`, `l_quantity`, `l_extendedprice`, `l_discount`, `l_tax`, `l_shipdate`, `l_commitdate`, `l_receiptdate`;
- a predicate should select 30-70% of the data, which requires `<value>` term of a predicate to be selected from the ranges provided in Table 4.1 and Table 4.2;
- the number of queries in the set and number of predicates in the queries are fixed;
- additionally, the percentage of duplicates in the list of simple predicates can be configured, since we would like to test, how presented horizontal fragmentation approaches will perform on the queries sharing some predicates;
- `<column>` term in the simple predicates should be evenly distributed over the set of above-mentioned numeric and date attributes of the `LINEITEM` table.

	orderkey	partkey	suppkey	linenumber	quantity	extendedprice	discount	tax
min (30%)	314691	59994	3007	2	16.00	22421.56	0.03	0.02
max (70%)	733478	140010	7005	4	36.00	51223.75	0.07	0.06

Table 4.1: Ranges of values used by the query generator for numeric fields

	shipdate	commitdate	receiptdate
min (30%)	1994-02-25	1994-02-25	1994-03-13
max (70%)	1996-10-12	1996-10-11	1996-10-27

Table 4.2: Ranges of values used by the query generator for date fields

For the experiments we generate the sets of queries with 10%, 20%, 30%, 40%, 50%, 60%, 70% of duplicates in simple predicates. For each of the duplicates percentage option we generate 10 sets of queries. Each set of queries contains 6 queries and each query has 3 simple predicates. An example of generated set of queries for 30% of duplicates is:

```
SELECT * FROM lineitem WHERE (l_receiptdate < '1994-08-05' AND
l_quantity >= 29.87735203286717 AND l_linenumber >= 2);

SELECT * FROM lineitem WHERE (l_receiptdate < '1994-08-05'
AND l_partkey < 95522 AND l_discount >= 0.05116648238656971);

SELECT * FROM lineitem WHERE (l_commitdate >= '1995-07-23' AND
l_tax >= 0.0314102265849505 AND l_orderkey < 381711);

SELECT * FROM lineitem WHERE (l_extendedprice < 44812.95404350133 AND
l_tax >= 0.026040578265457394 AND l_shipdate < '1995-07-19');

SELECT * FROM lineitem WHERE (l_receiptdate < '1994-08-05' AND
l_shipdate < '1995-07-19' AND l_suppkey < 4079);

SELECT * FROM lineitem WHERE (l_extendedprice < 44812.95404350133 AND
l_partkey >= 67271 AND l_shipdate < '1995-07-19');
```

4.4 Algorithms settings

General settings: We set k (number of horizontal fragments) for all of the discussed solutions to 4.

Deep-RL experiment design

The Deep-RL solution is tested in two modes: with a fixed query workload and generalized over the whole testbench provided by the query generator.

The first test serves to prove that the design of the RL environment suits the task of finding the optimal horizontal fragmentation and that the model can mine the data from a fixed workload. The design of this experiment is straightforward: the specified query workload is passed to runner, which then starts to continuously train and evaluate the specifies agent on it. Each run consists of a fixed number of iterations, which represent a fixed number of training steps followed by model evaluation. In the experiments with fixed workload, the training and evaluation happen on the same batch of queries.

The generalization test implies that training and evaluation follow different rules on picking queries. In our generalization design we evaluate the agent over all queries provided by testbench to track model performance across all duplicate buckets. The queries for training are picked from a pool using weighted random choice. Each query batch is assigned weight, which is inversely proportional to the recorded performance of the agent on it. This ensures that agents train more on data they have not yet managed to master.

Hyper-parameters for DQN agents

Dopamine provides a number of configurable parameters for the DQN agent family. In most experiments the majority of them is kept in their default values. In Table 4.3 we provide a brief description of the key parameters and the values we set them to, in order to give insights into the learning process and ensure reproducibility of our results.

Parameter	**Description**	**Value**
training_steps	The maximum length of a single training phase of iteration	1000
evaluation_steps	The maximum length of evaluation phase of iteration. This value is a soft limit, meaning that once the episode has started, no checks are performed until next episode starts	1
num_iterations	Number of iterations (training + evaluation phases)	Varies depending on the experiment
gamma	Discount factor γ for future rewards	0.99
epsilon_train	Epsilon value for training phase at the end of epsilon decay	0.01

epsilon_decay_period	Number of steps in which epsilon reaches epsilon_train	num_itearations * training_steps * 0.95
target_update_period	Update period of target NN	200
update_horizon	Number of steps in which the model performs its Q-values update	Varies depending on the experiment
min_replay_history	Minimum number of steps stored in the replay memory for agent to start updating target NN	2000
replay_capacity	Maximum number of steps stored in replay memory	1000000
batch_size	Number of steps sampled from replay memory each update	32
learning_rate	Learning rate of gradient descent	0.00005
optimizer	Optimization algorithm used to update NN towards minimizing objective function	Adam [71]

Table 4.3: Parameters of Dopamine agents of DQN family

4.5 Summary

In this chapter our experimental setup is discussed. We disclose hardware and software characteristics of the experimental environment, describe the dataset and the workloads used in the experiments. We also discuss important configuration parameters of the proposed horizontal fragmentation solutions.

5. Evaluation and Results

In this chapter we compare a classic horizontal fragmentation algorithm designed by Zhang and Orlowska [1] and our proposed clustering-based and RL-based horizontal fragmentation solutions. We consider different parameters, which can influence the results and we analyze the quality of the fragmentation schemes generated by the algorithms.

We structure the chapter as follows:

- In Section 5.1 we formulate the research questions we intend to answer with evaluations.
- In Section 5.2 we document the evaluation results of clustering-based solution. We provide comparison of the solution's performance with different configuration parameters.
- In Section 5.3 we provide the evaluation details of the Deep-RL based solution. We document its performance in different training modes and discuss the impact of hyperparameters on convergence and performance of Deep-RL models.
- In Section 5.4 we provide the final evaluation, comparing the both clustering-based and Deep-RL-based algorithms with each other as well as with a classical baseline.

5.1 Research questions

In this section we refine some of the research questions presented in Section 1.2 that will be answered in this chapter:

1. To what extent can the clustering-based fragmentation approach reduce the query execution cost compared to the classical fragmentation approach?

2. To what extent can the cost model usage improve the fragmentation schemes generated by the clustering-based fragmentation approach?

3. Which similarity measure and linkage criterion for the clustering-based solution result in the lowest execution costs?

4. What is the training cost and impact of parameters for an RL-based approach?

5. How does the choice of Deep-RL model affect the convergence of the RL-solution?

6. What are the optimal parameters for a representative evaluation that would capture the best performance of both RL- and UL-based approaches?

7. How well do proposed ML-based approaches perform compared to each other and to state-of-the-art horizontal fragmentation algorithms?

5.2 Clustering-based solution

There are numerous internal and external evaluation criteria for a clustering algorithm, including the following: SSQ, Davies–Bouldin index, Silhouette coefficient, Adjusted Rand Index, Normalized Mutual Information, Purity [29]. However, we cannot fully rely on these criteria, as we have incorporated a cost model into our approach to control and guide the clustering process. Moreover, we would like to compare our clustering algorithm with the classical horizontal fragmentation algorithm and with our fully cost-model-driven RL-based approach. Therefore, in this chapter we compare our solutions based on the estimated execution cost of the queries on a fragmented table.

To design meaningful tests we need to define parameters, which might influence the results generated by the clustering-based horizontal fragmentation algorithm. In section Section 3.5 we mentioned that the fragmentation scheme obtained by applying the algorithm can differ depending on the linkage criterion and similarity measure used, so in the experiments we test all relevant combinations of them:

- Single-linkage criterion(SL) & Euclidean distance;
- Single-linkage criterion(SL) & Squared Euclidean distance;
- Single-linkage criterion(SL) & Manhattan distance;
- Single-linkage criterion(SL) & Maximum distance;
- Complete-linkage criterion(CL) & Euclidean distance;
- Complete-linkage criterion(CL) & Squared Euclidean distance;
- Complete-linkage criterion(CL) & Manhattan distance;

- Complete-linkage criterion(CL) & Maximum distance;
- Average-linkage criterion(AL) & Euclidean distance;
- Average-linkage criterion(AL) & Squared Euclidean distance;
- Average-linkage criterion(AL) & Manhattan distance;
- Average-linkage criterion(AL) & Maximum distance;
- Centroid-linkage criterion(CentrL) & Euclidean distance;
- Penalty-based method.

Another parameter that we would like to consider is the cost model usage. That was one of the main requirements for the clustering-based fragmentation solution. The reason for that is obvious: the cost model usage might significantly improve the quality of the results. However, it should be possible to run the algorithm without a cost model included. This feature should always be optional, because it is not always an easy task to choose a suitable cost model. Moreover, the selected cost model might be a performance bottleneck. If a cost model takes a lot of time, it is important to configure the algorithm in such a way that the algorithm produces good enough results within a reasonable amount of time.

In our case this is definitely an important factor to consider. With Single-linkage criterion and Euclidean distance our clustering-based fragmentation algorithm spends 93.2% of the time on estimating the execution costs using the cost model (Figure 5.1). The approach based on the algorithm designed by Zhang and Orlowska [1] behaves similarly and spends almost 100% of the time on the cost model usage (Figure 5.2). Therefore, we use the number of cost model calls not only to compare different similarity measures and linkage criteria with each other, but also to compare our clustering-based fragmentation solution with the classical fragmentation approach developed by Zhang and Orlowska.

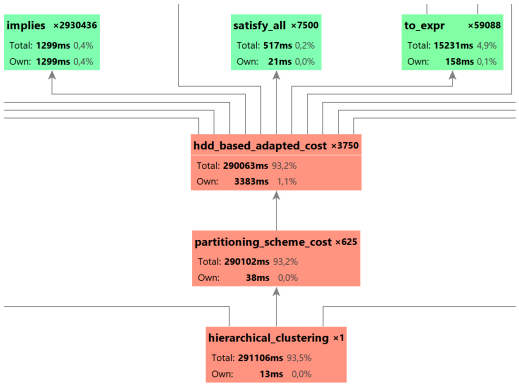


Figure 5.1: Profiling results (the clustering-based solution)

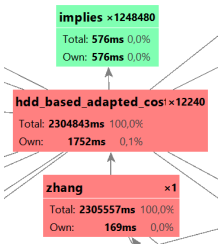


Figure 5.2: Profiling results (the classical fragmentation approach)

5.2.1 No cost model included

To answer the question of how the choice of a similarity measure and linkage criterion influences the quality of the generated fragmentation schemes, we first need to consider them "in isolation", i.e. with no cost model included into the clustering process.

We expect the Average-linkage and Penalty-based method to outperform other linkage criteria, because the Average-linkage criterion is generally considered to be very effective, especially when there is no information about the shape of the clusters in the input data. In Average-linkage approach each element of a cluster contributes to the calculation of the distance between the clusters. The Penalty-based method was developed specifically for the horizontal fragmentation task and we expect it to generate the best results among all the linkage criteria. The experiments were performed on the ten sets of queries generated by the query generator with 10% of duplicates in the input predicates. The results presented in Figure 5.3 are average values of the results generated from each query workload.

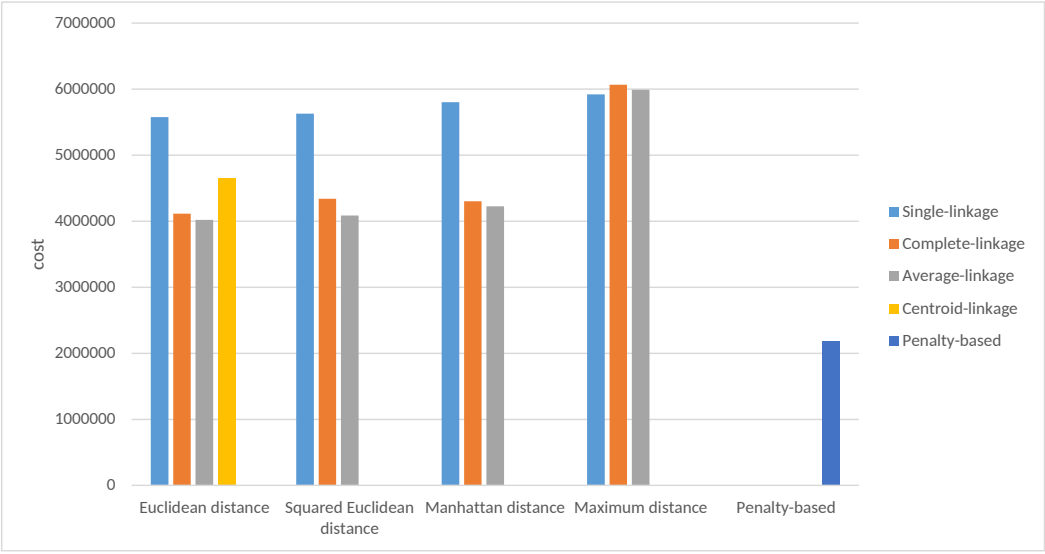


Figure 5.3: Clustering-based solution without cost model usage (execution costs are measured in tuples fetched)

Analysis

- *Single-linkage criterion generally leads to the worst results (5,578,129 vs 4,115,017 with Complete-linkage and 4,020,392 with Average-linkage using Euclidean distance; 5,629,254 vs 4,340,258 with Complete-linkage and 4,087,210 with Average-linkage using Squared Euclidean distance; 5,803,449 vs 4,302,953 with Complete-linkage and 4,224,468 with Average-linkage using Manhattan distance).*

  The reason for this could be that the Single-linkage criterion can cause a chaining effect (or chaining phenomenon). It means that if there are two clusters close to each other, they might end up being merged together even though many of their elements are very far from each other.

- *Clustering using the Average-linkage generally produces better results than the one using Complete-linkage.*

  The nature of the Complete-linkage causes the clusters generated using this criterion to be compact, but it makes it very sensitive to outliers.

- *In average the Penalty-based method results in 2.3 times better solutions than the other methods (including Average-linkage and Centroid-linkage).*

  Average-linkage and Centroid-linkage might provide better results in terms of internal and external validity, but since our Penalty-based method was specifically designed to meet the concept of horizontal fragmentation and our final goal is to generate the best possible fragmentation schemes in terms of the execution cost, it is clear that the Penalty-based method will outperform other classical linkage methods.

- *Maximum distance leads to the worst results in comparison to the other similarity measures.*

  There is not much difference between using similarity measures like Euclidean, Squared Euclidean or Manhattan (average costs of 4,591,382, 4,685,574 and 4,776,956 respectively) but Maximum distance turned out to be the worst fit for our task with an average cost of 5,992,403. The reason behind this is that with the selected input data representation the Maximum distance between two points will always be either a 0 or 1. As a result, there are a lot of pairs of clusters at each iteration with the same distances to each other and with no cost model included a pair of clusters to merge will be selected randomly. Obviously, when using a hierarchical clustering algorithm, where the changes made once cannot be undone, random selection is not the best strategy to follow.

5.2.2 With the cost model included

As mentioned earlier, along with the execution cost the number of cost model calls is a very important factor for us. Our general expectation would be that the more the

algorithm uses the cost model, the better results it will generate. The number of cost model calls depends in turn on the diversity of distance values that a similarity measure and a linkage criterion can yield. But of course different similarity measures and linkage criteria will lead to different results even with the same number of cost model calls, as shown in the previous section. In this experiment we use the same input data and test the same linkage criteria and similarity measures, but with the cost model included. The average execution cost of the queries is presented in Figure 5.4 and the average number of the cost model calls is shown in Figure 5.5.

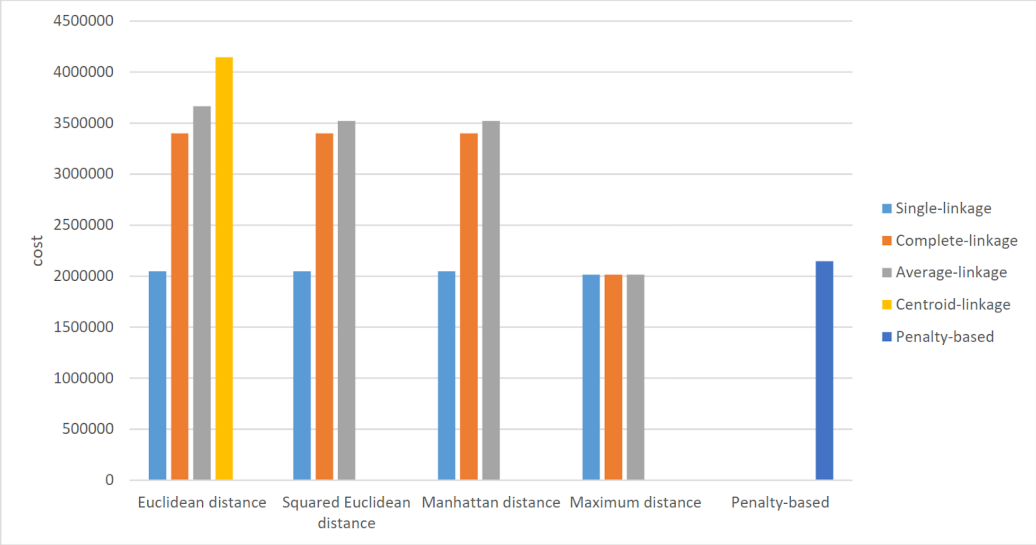


Figure 5.4: Clustering-based solution with the cost model usage (execution costs are measured in tuples fetched)

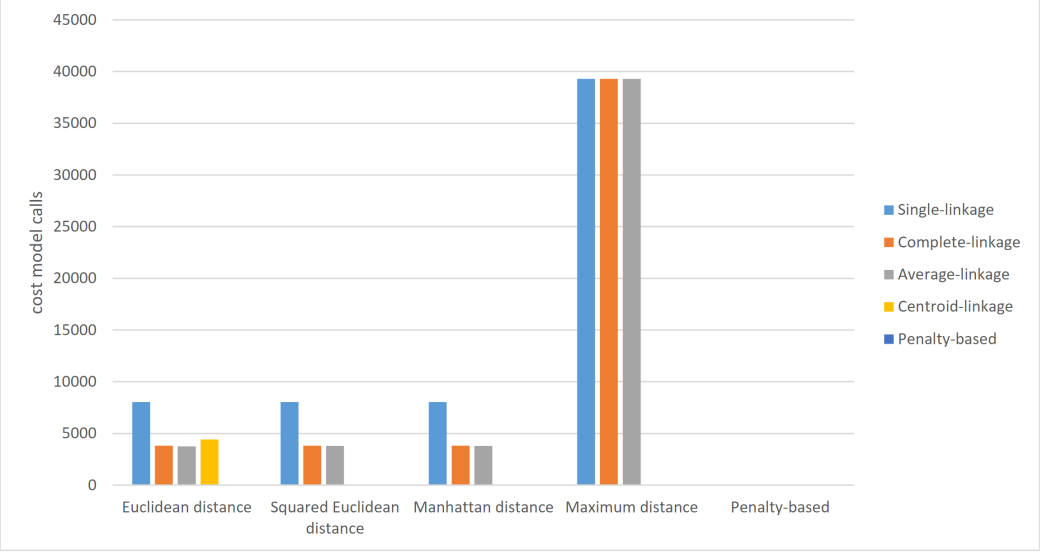


Figure 5.5: Clustering-based solution with the cost model usage (number of cost model calls)

Analysis

The following conclusions can be drawn from the results of this experiment:

- *The best fragmentation schemes were generated using the Maximum distance.*

  In contrast to the experiments conducted in the previous section, fragmentation schemes generated using the cost model and the Maximum distance lead to the lowest execution costs of the queries (2,013,827), which was absolutely predictable considering how many cost model calls it needs (39,307) to achieve such results. The clustering algorithm with the Single-linkage produces results not much worse than with the Maximum distance (2,047,838) but using much less cost model calls (8039). A great alternative to using Maximum distance is to use the Penalty-based method, since it also produces very good results (2,145,374) and calls the cost model only 9 times.

- *Single-linkage is proportionally almost as good as Complete- and Average-linkages.*

  The execution costs of the queries on the fragmentation schemes generated using Single-linkage are almost 2 times lower than those generated using Complete- and Average-linkages, but the number of cost model calls is in turn twice as high. The reason why the algorithm calls the cost model so many times with the Single-linkage lies in the fact that the input data for our clustering algorithm generally consists of all possible combinations of Boolean values and the number of combinations, which differ only in one value, is higher than the number of combinations, which differ in all the values. For instance, with the Single-linkage selected and 4 queries used maximum number of cluster pairs with the same distances to each other is 32 at the first iteration, whereas with the Complete-linkage selected this number would be 8.

- *Centroid-linkage is outperformed by Average- and Complete-linkages.*

  The execution costs of the queries on the fragmentation schemes generated using Centroid-linkage (4,145,271) is higher, than on the fragmentation schemes generated using Average- and Complete-linkages and considering that the number of cost model calls when using Centroid-linkage (4,416) is also higher than the number of cost model calls when using Average- and Complete-linkages we can conclude that the concept of centroids does not fit well to the task and the selected input data representation.

- *The Penalty-based method seems to be one of the best methods for calculating the distances between clusters.*

  Besides generating very good fragmentation schemes, the algorithm needs an incredibly small number of cost model calls. This is due to the fact that this method incorporates information about the number of rows in the atomic fragments to estimate the execution cost, which leads to the quite large value space of the distance function. That means that most of the time the method can rely on

its own estimation of the distance to distinguish between different cluster pairs instead of calling the cost function for that.

Using the results obtained in this experiment we can choose the best similarity measure and linkage criterion pairs for our next experiments. Figure 5.6 depicts the Pareto-optimal solutions for the multi-objective optimization problem, which involves two objectives: minimization of the execution costs and minimization of the number of cost model calls.

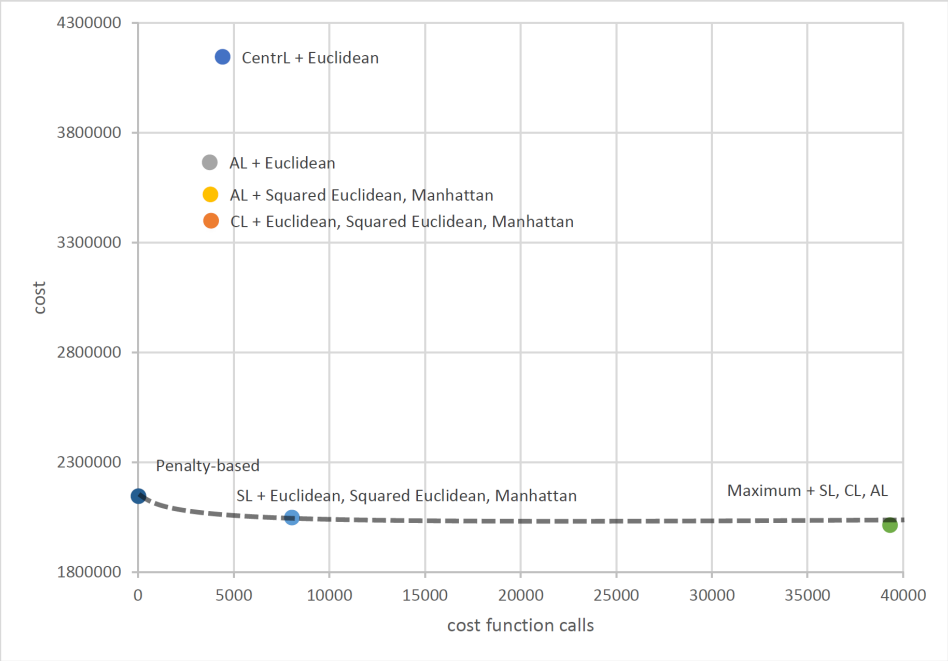


Figure 5.6: Pareto-optimal solutions (execution costs are measured in tuples fetched)

Based on the results presented in Figure 5.6 the following methods were selected for the further experiments:

- Maximum distance & Complete-linkage;
- Euclidean distance & Single-linkage;
- Penalty-based method.

5.3 Deep-RL-based solution

In this section we intend to answer the research question of how different different Deep-RL algorithms tackle solving the task of finding an optimal horizontal fragmentation; and what are the key factors that influence their convergence and performance. As

discussed in Section 3.6, we use Dopamine framework and the models bundled with it to evaluate our Deep-RL approach.

In our evaluation we use the following models from the DQN algorithm family:

- Rainbow DQN
- Implicit Quantile (IQN)

We decided to skip testing with DQN algorithm itself, as its extensions (mentioned above) tend to outperform it on most benchmarks.

5.3.1 Convergence in the case of fixed workload

We start our experiments by ensuring convergence for a simple case of fixed query workload. We expect all agents to converge very early for such trivial task; therefore, we set maximum number of iterations to 100.

As mentioned in chapter Chapter 4, the testbench for our solutions provides 7 distinct query buckets, with different percentage of predicate duplicates. We provide a results of execution costs for single query from each bucket to illustrate convergence.

As we can see, all agents converge very quickly in first couple of iterations (10000 steps). We expect fast convergence due to the fact that the task is pretty straightforward. This test shows that the Dopamine models can successfully mine the features from data representation we designed.

Figure 5.7 also shows that the Rainbow agent proves to be slightly unstable on some workloads, with the execution cost oscillating after convergence.

We also have ran the tests with variable to evaluate influence of update horizon (UH) parameter on speed of convergence, but the models converge too fast to see significant difference. UH defines a number of steps in which the model performs its Q-values update.

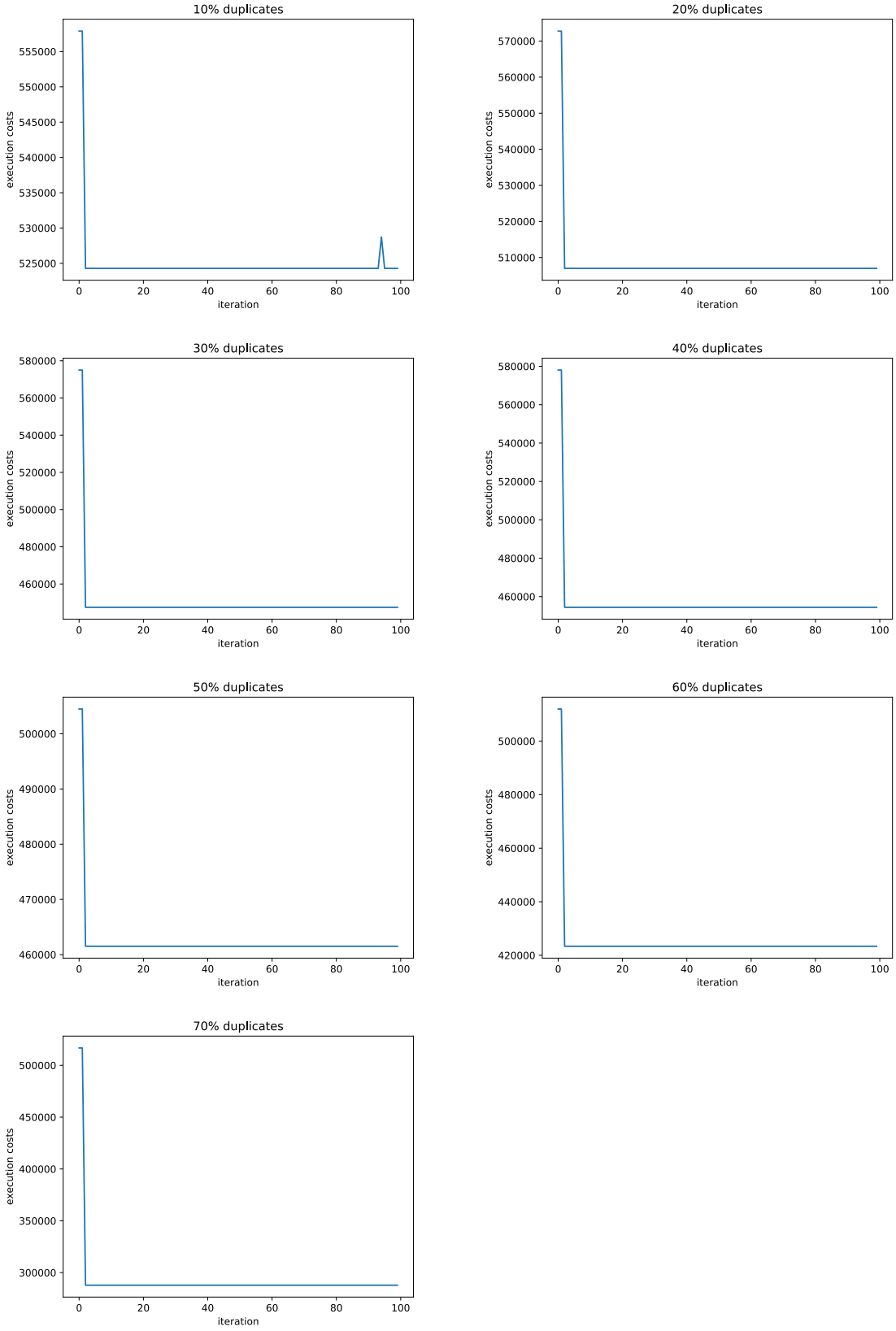


Figure 5.7: Rainbow DQN in the case of fixed workload (execution costs are measured in tuples fetched)

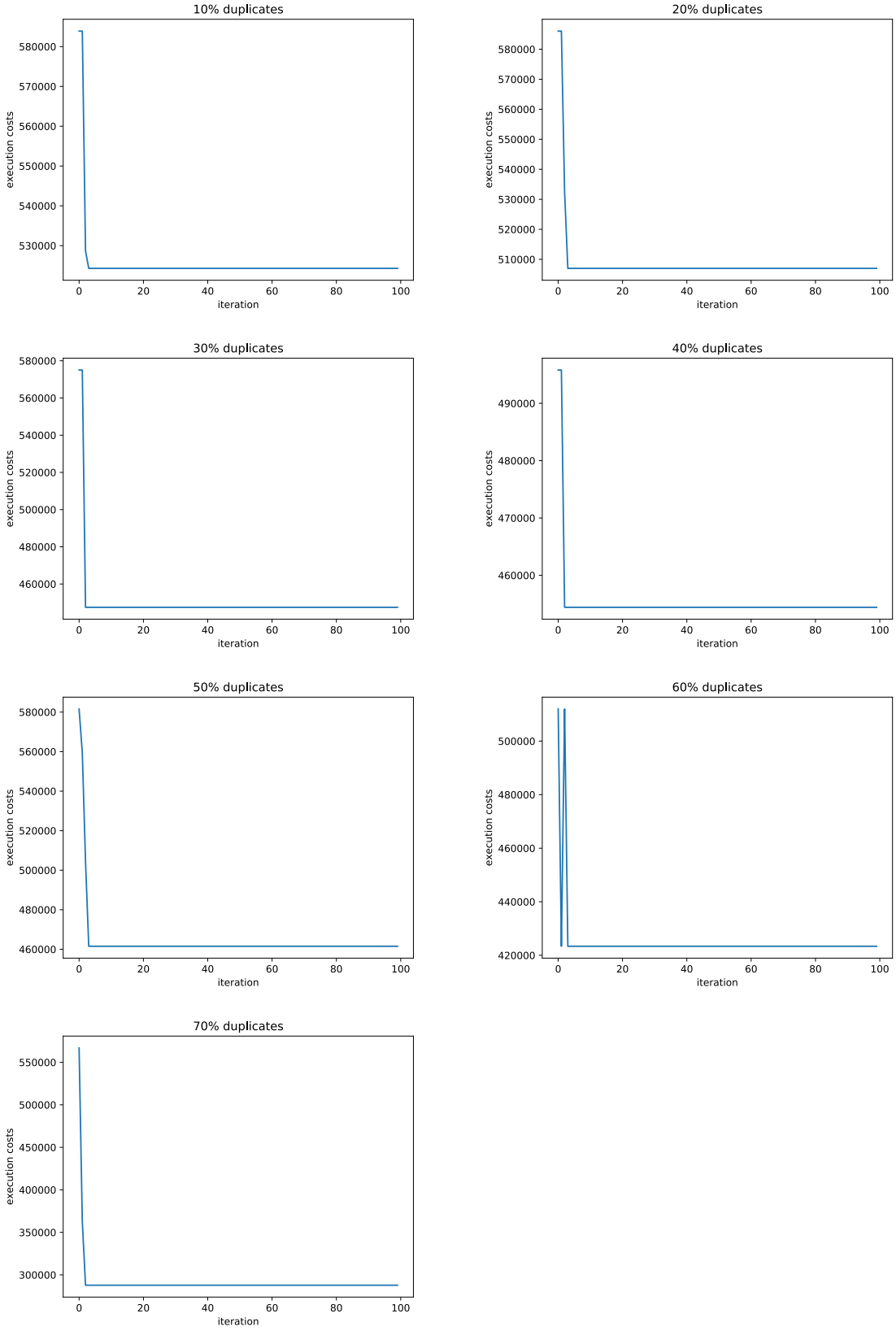


Figure 5.8: Implicit Quantile in the case of fixed workload (execution costs are measured in tuples fetched)

5.3.2 Convergence in the case of generalized workload

As mentioned in chapter Chapter 4, the testbench for our solutions provides 7 distinct query buckets, with different percentage of predicate duplicates. We provide a results of average execution costs for each bucket to illustrate convergence.

For this experiment we fix all of the parameters of Dopamine agents except for the update horizon (UH). We test the following model-UH configurations:

- Rainbow DQN + UH=1;
- Rainbow DQN + UH=2;
- Rainbow DQN + UH=4;
- Implicit Quantile + UH=1;
- Implicit Quantile + UH=2;
- Implicit Quantile + UH=4;

The models are trained for 2000 iterations on workloads, which are randomly chosen from the testbench as described in Section 4.4. We track the query execution performance for all queries from the testbench, grouping it by predicate duplicates percentage. The Figure 5.9 provides the results of the average cost for each duplicates bucket on each iteration of the training of the Rainbow agent.

Both agents tend to converge during the training. The graphs show that results of both agents oscillate even when it has nearly converged, signalling that the randomly selected environments skew their predictions at each training iteration. However, oscillations decrease with time, proving that the agents can learn the general patterns from the data encoded by our proposed environment.

The experiments show that the Rainbow agent manages to converge faster than Implicit Quantile in generalization scenario and tends to deliver better results. The change of UH parameter impacts agent's results differently. While the increase of UH for Rainbow agent results in slightly faster convergence for high duplicates percentage, the increase of it for Implicit Quantile agent does not yield noticeable improvement, and in some cases even slows the convergence down.

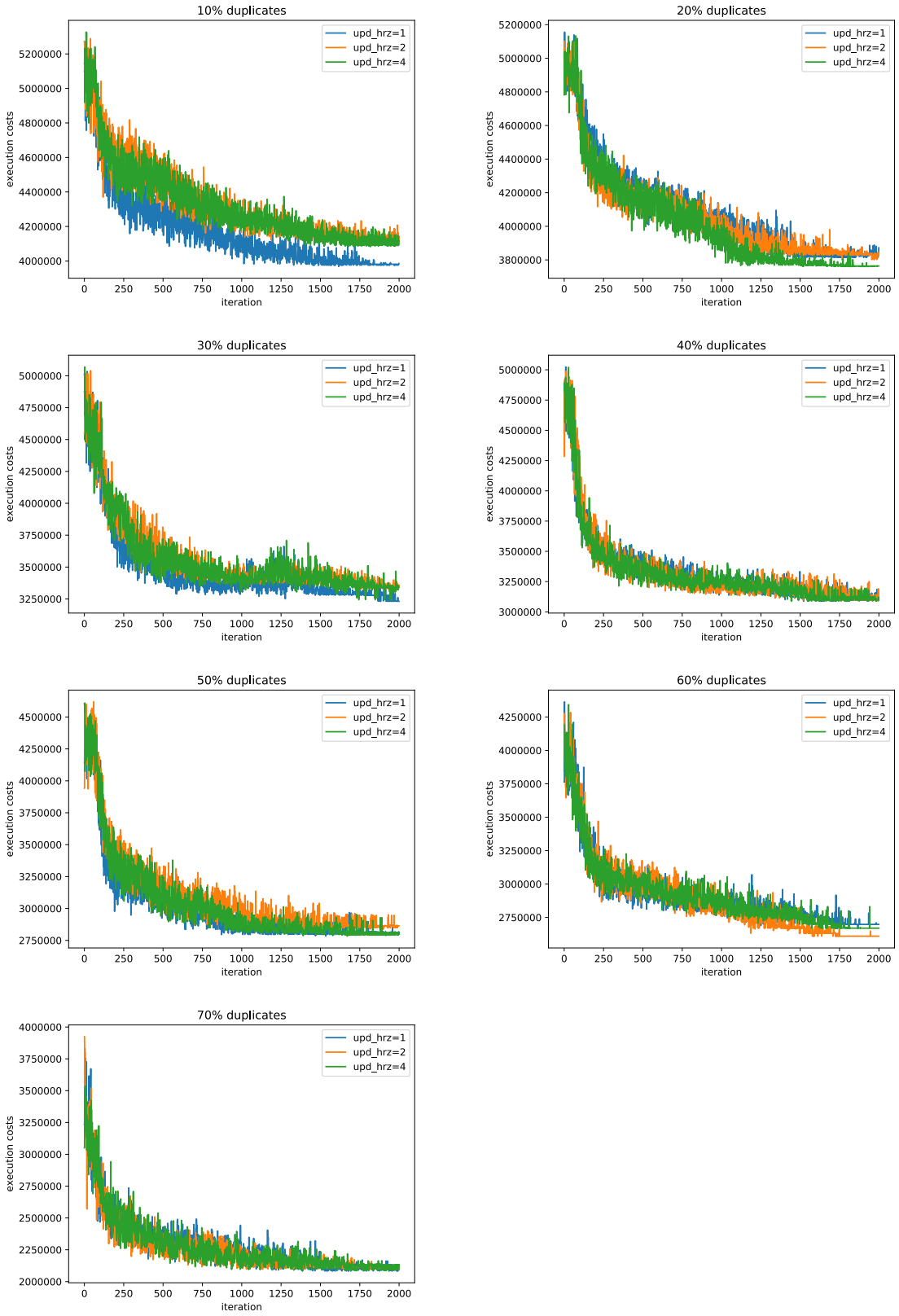


Figure 5.9: Rainbow DQN in the case of generalized workload (execution costs are measured in tuples fetched)

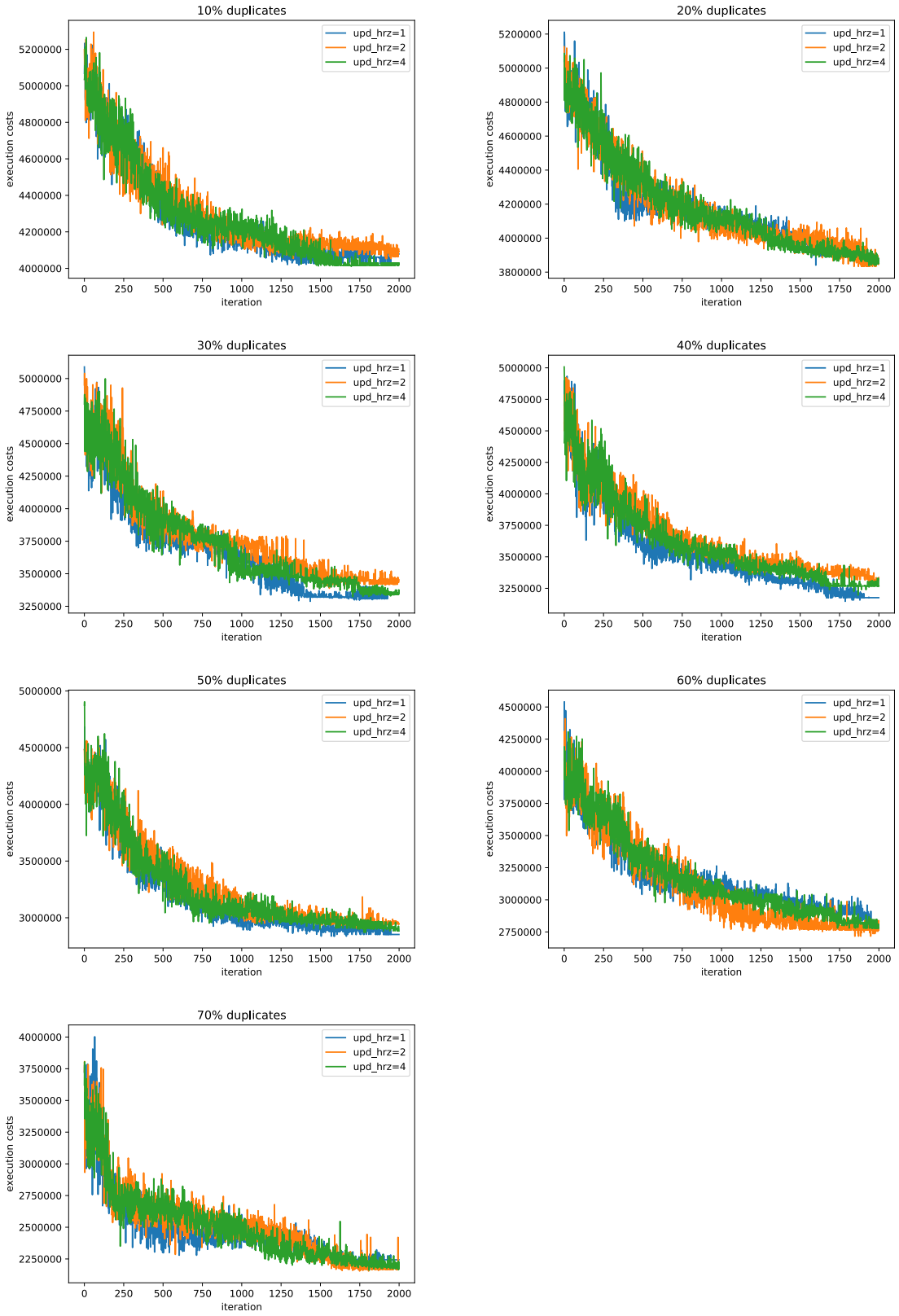


Figure 5.10: Implicit Quantile in the case of generalized workload (execution costs are measured in tuples fetched)

Since the results produced by Implicit Quantile and Rainbow are slightly different after full training period, we decided to include both of them in final evaluation.

5.4 Comparison of the solutions

In the final experiment we compare three horizontal fragmentation solutions discussed in this thesis:

- classical affinity-based algorithm proposed by Zhang and Orlowska [1];
- clustering-based solution with the distance calculation methods selected in Section 5.2;
- RL-based approach.

We surmise, that the results obtained by using the classical fragmentation approach might differ depending on the percentage of duplicates, that the queries have in predicates. So for this experiment we use sets of queries with 10%, 20%, 30%, 40%, 50%, 60% and 70% of duplicates (10 sets of queries each). We compare average execution costs of the queries on the fragmentation schemes generated by all 3 approaches and we also provide the data on the number of cost model calls, which was needed to generate the results using the classical algorithm and the clustering-based algorithm.

We do not evaluate the number of cost model calls used by RL-based solution, because the fully trained agent does not require the cost model, and the usage of the cost model in training accounts for *all* the possible workloads agent can process thereafter (even previously unseen ones). Instead, we provide inference times for the agents.

5.4.1 Quality of the results

In this section we compare the quality of the fragmentation schemes generated by the proposed solutions using the cost model described in Section 3.3 (Figure 5.11).

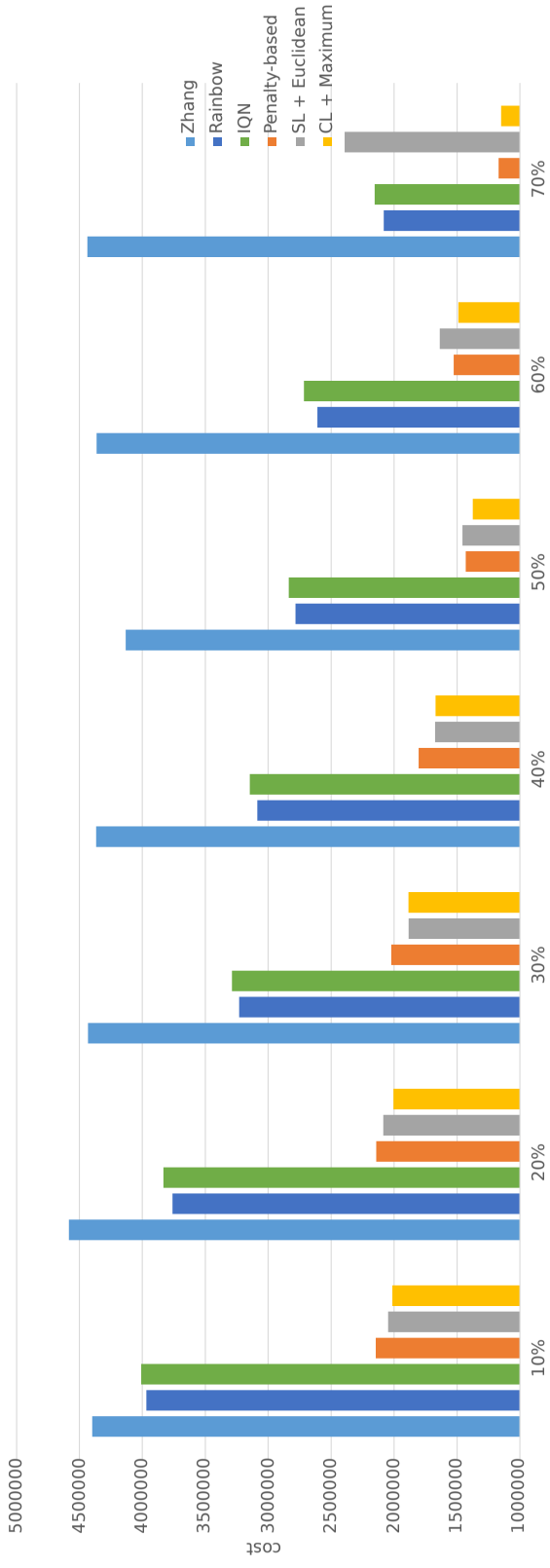


Figure 5.11: Quality of the fragmentation schemes generated by the horizontal fragmentation solutions (execution costs are measured in tuples fetched)

Analysis

- *Fragmentation schemes generated using Complete-linkage and Maximum distance have the lowest average cost.*

  Complete-linkage in combination with the Maximum distance takes first place with the average result of 1,655,485, Penalty-based method takes second place with the result of 1,748,916.

- *The impact of the percentage of duplicates in the queries.*

  The query execution performance for Deep-RL tends to improve as the duplicates percentage grows. This was expected, as the increase in the amount of duplicates makes its task easier. Having the same predicates in multiple queries means that fragmenting the table on those predicates improves the performance on more than one query. However, both the clustering-based and the classical algorithm proposed by Zhang et al. use different fragmentation strategies, and quality of their results does not seem to correlate with the percentage of duplicates.

- *General quality of the results.*

  The results of the solutions seem to follow a certain trend: clustering-based solution provides the results of the best quality (generally 2 times better than the results of the RL-based solution). Both ML solutions outperform the baseline.

To analyze the behaviour of the solutions on different input data sets it is not enough to compare the quality of the generated fragmentation schemes. That is why in the next section we compare the average number of the cost model calls required by the algorithms on the same input data.

5.4.2 Number of cost model calls

Since the data on the number of cost model calls might be very important for a designer to make a choice between different fragmentation solutions, in Figure 5.12 we illustrate the dependency between percentage of predicate duplicates in the queries and the number of cost model calls for the discussed horizontal fragmentation approaches.

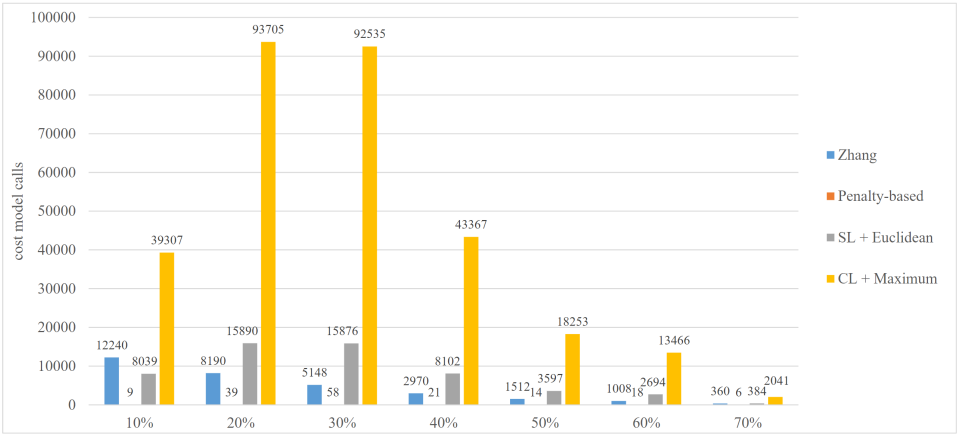


Figure 5.12: Number of the cost model calls of the clustering-based and the classical fragmentation solutions

Analysis

- *The number of cost model calls in the classical fragmentation algorithm depends on the percentage of predicate duplicates in the queries.*

  As far as the algorithm proposed by Zhang and Orlowska is concerned, there is a tendency for the number of cost model calls to be reduced with the increase in the number of predicate duplicates in the queries. The reason for that lies in reduction of the PA matrix, since the matrix is built from the set of unique predicates.

- *The number of cost model calls in the clustering-based fragmentation algorithm does not directly depend on the percentage of predicate duplicates in the queries.*

  As for the clustering-based approach, there is no clear correlation between the percentage of the duplicates and the number of cost model calls. But it seems like with all of the distance calculation methods that we consider (Penalty-based method, Single-linkage with Euclidean distance and Complete-linkage with Maximum distance) the number of cost model calls depends on the average number of initial atomic fragments, that changes as shown in Table 5.5.

percentage of the duplicates	10	20	30	40	50	60	70
number of cost model calls	32	42	42	31	22	21	11

Table 5.5: Average number of initial atomic fragments

- *Clustering-based fragmentation algorithm with the Penalty-based method requires the least number of the cost model calls.*

The average number of the cost model calls required by the method is 23, which, in comparison to the other methods (7,797 by Single-linkage and Euclidean distance; 43,239 by Complete-linkage and Maximum distance and 4,489 by the classical algorithm), is a pretty good result.

5.4.3 Inference times

Unlike the algorithm proposed by Zhang and Orlowska and the clustering-based solution, proposed in this thesis, the Deep-RL-based solution does not require the cost model usage once it is fully trained. This section provides the comparison between the time needed for each agent to produce fragmentation for a single workload.

Rainbow	Implicit Quantile
1.1343s	1.0309s

Table 5.6: Inference times for Rainbow and Implicit Quantile

Table 5.7 shows the elapsed times for a single cost model usage. Given that the clustering-based solution and the algorithm proposed by Zhang and Orlowska use the cost model multiple thousand times to process a single workload (presented in Section 4.3), a fully-trained Deep-RL-based solution ends up several orders of magnitude faster.

Zhang	clustering-based	Deep-RL-based
0.0496s	0.0154s	0.0007s

Table 5.7: Average cost model use times

5.5 Summary

In this chapter we provide the results of the evaluation of both of our proposed solutions. We evaluate the solutions separately and then proceed with the comparison between them and the baseline. We wrap this chapter up with comparison of the cost model usage between the baseline and the clustering-based solution, and the inference times for the Deep-RL-based solution.

6. Related work and Future Directions

In this chapter we review papers that use ML-based approaches for solving related database physical design problems. We compare our solutions with those presented in the literature and spot some ideas, which can be integrated into our solutions in the future.

We structure the chapter as follows:

- In Section 6.1 we discuss papers where clustering algorithms are used for solving the issues related to the horizontal fragmentation problem.
- In Section 6.2 we review RL-based solutions to the various physical design problems.
- In Section 6.3 we analyze papers that study usage of other optimization algorithms for the database fragmentation.
- We summarize the whole chapter in Section 6.4

6.1 Clustering for physical design problems

Clustering algorithms can be applied to various physical design problems. Papers that are concerned with the task being solved in this thesis, were in detail described in Section 2.3.2.2. In this section we compare our work with the papers that are devoted to problems slightly different from ours and use clustering-based algorithms to solve them.

For instance, in related works [72, 73] an algorithm for incremental horizontal fragmentation was proposed. Authors concentrate on the problem of an evolving user applications set and they claim that rerunning the horizontal fragmentation algorithm might be an obvious but very inefficient solution. That is why they extend their clustering-based

fragmentation algorithm developed before [32] by adapting it to an evolving set of queries. Initially, fragments are generated by applying k-means over object-condition vectors. When a new set of queries arrives, k-means Core-Based Incremental Clustering (CBIC) algorithm is used to adapt the fragmentation scheme obtained in the previous step. The algorithm selects a set of objects, which after feature extension remain close to the centroid of a cluster. These objects are called a core of the cluster and they become initial clusters for the iterative fragmentation process. K-means and CBIC clustering results are compared using the core stability factor measure and the fragmentation results are compared using a cost function that considers the local irrelevant access and remote relevant access costs. The CBIC algorithm is proven to be effective for horizontal fragmentation in an evolving environment, except for cases when new user queries require a lot of changes to the initial fragmentation scheme.

Using similar concepts of the core-based clustering authors introduce a hierarchical adaptive clustering algorithm [74], which is an extension of the classical hierarchical clustering algorithm used in their previous works. Authors claim that it can also be effectively used for horizontal re-fragmentation. Although our clustering-based fragmentation solution does not focus on the problem of the evolving user queries, the results shown in these papers seem to be very promising and the idea of incremental adaptive clustering can be integrated into our clustering-based approach in the future. This would contribute to making our solution more competitive with the feature learning of the generalized RL-approach.

Horizontal fragmentation can also be performed during the initial database design phase. Ramachandran et al. [75] show how clustering algorithms can be applied to this task. The algorithm proposed in the paper considers relationships between the data being fragmented, because users usually select related (similar) data and storing such data in the same fragment can significantly improve query response time. In contrast to our clustering-based horizontal fragmentation solution, the algorithm described in the paper does not consider empirical query patterns. However, the clustering approach they use has something in common with our approach. The first step is to group similar objects together. Then from each group a prototype (representative) is selected. These cluster representatives are used for calculating the similarity between clusters and merging the similar clusters together till the specified amount of fragments is reached. Authors perform experiments using a PostgreSQL database and compare the fragmentation scheme generated by their algorithm with a randomly generated fragmentation scheme. Although the experiments show significant performance improvements, in order to talk about the relevance of applying this approach, the results generated by it must be compared with those generated by existing horizontal fragmentation algorithms.

Amina et al. [76] address the issue of very large workloads in the horizontal fragmentation problem. Since this constitutes an NP-complete problem, traditional approaches do not perform well when the workload is very large. For solving this issue authors propose a method which consists of two phases: classification of the queries in the workload and election of a query from each generated class (Figure 6.1). For the classification phase authors use k-means algorithm, which groups queries according to the data they

reference. Authors validate their solution in Oracle and show that the method not only generates a horizontal fragmentation scheme much faster than the existing algorithms, but it also produces results with the similar or even better quality. As of now, neither of the proposed solutions in our thesis works well on large workloads. In the future the method of Amina et al. can be integrated into our solutions to improve their scalability and reduce their computational complexity.

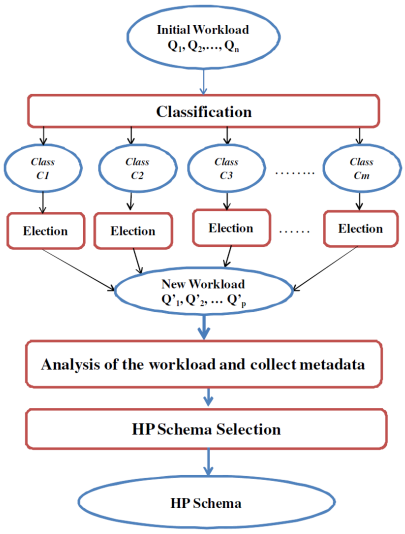


Figure 6.1: Horizontal fragmentation approach proposed by Amina et al. [76]

6.2 RL for physical design problems

In this section we provide an overview of the papers that apply RL to tasks of physical database design and database optimization problems, such as join-order enumeration, vertical and horizontal fragmentation, etc.

Campero Durand et al. [77] propose the GridFormation framework for formulating database fragmentation problems in terms of RL, emphasizing the online self-managing capabilities of RL-based solutions. This paper presents an early implementation of the grid environment, deconstructing the database fragmentation problem into a sequence of steps to be taken by the RL agent. The authors compare the convergence of Deep Q-learning based approach against straightforward Q-learning implementation; however, the proposed framework lacks the integration with the database.

In [78] authors expand on the ideas of GridFormation, applying the Deep-RL learning to the vertical fragmentation problem. The paper presents an approach of re-framing the problem of finding the right vertical fragmentation in RL terms. The authors provide early evaluation results, testing their solution against single workload-table pair as well as the generalized case of random workloads with fixed table. The evaluations are performed using tables and queries from the TPC-H benchmark. The authors report that their solution performs well in the case of the fixed workload; however, the case of

generalizing to random query workload proves tricky for the model to grasp and requires further work.

Hilprecht et al. propose [79] a fragmentation advisor based on Deep-RL, capable of managing multiple tables at once. The proposed solution operates using hash partitioning, fragmenting relations horizontally on the selected attribute. The solutions also manages replication and hybrid horizontal fragmentation using co-partitions for the tables, partitioned on the same attribute. The Deep-RL agent at the core of the approach proposed by Hilprecht et al. can choose table and attribute for fragmentation; however, fragmentation details beyond that are hidden from the model. The paper also describes a training process for Deep-RL agent, aimed at deployment of said agent as an automated fragmentation manager. The overview of the approach proposed by Hilprecht et al. is shown in Figure 6.2:

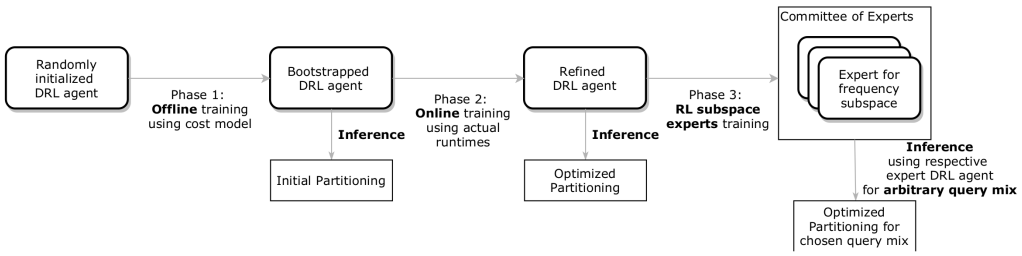


Figure 6.2: Overview of DRL-based approach to Learn a Partitioning Advisor [79]

The authors also provide comprehensive evaluation and analysis of the performance of agent in both pre-trained and online-learning modes. The paper provides the results of the workload adaptivity tests, measuring how the solution performs when exposed to the queries it was not trained with.

Marcus et al proposed a solution for optimizing the order of join operations during query execution using Deep-RL, called ReJOIN [80]. The authors adopt a common cost-based approach for query optimization, encoding the information about the current join ordering in a state vector, which is then used as a input of the NN. Figure 6.3 shows the outline of the ReJOIN framework.

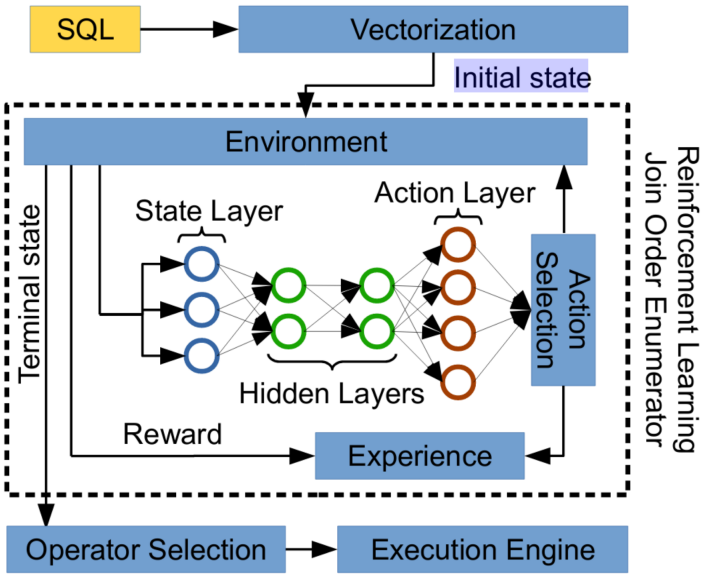


Figure 6.3: ReJOIN framework [80]

The authors use a so called Join Order Benchmark and IMDB dataset to evaluate their solution. The paper provides the result that confirm ReJOIN outperforming PostgreSQL optimizer by 20% in terms of join-order cost.

In contrast to most related work applying RL to database tasks, we use standard implementations of the agents, supported by an existing DRL framework. We also provide a design for the horizontal fragmentation task that differs substantially from those in the literature, since we do not consider hash partitioning. In future work we could consider, as does Hillprecht et al., pre-training and creation of expert agents.

6.3 Other optimization algorithms for physical design problems

Another type of ML algorithms that can be used for solving such computationally hard problems of physical design as fragmentation or allocation is genetic algorithms. These population-based metaheuristic algorithms are known to be efficient for optimization and search problems.

Gorla et al. [81] apply genetic algorithms to the hybrid fragmentation task. They use a genetic algorithm to iteratively fragment database tables into vertical and horizontal fragments until a termination criterion is satisfied. The genetic algorithm takes a transaction profile as input, which consists of an attribute usage matrix and a tuple usage matrix. Such tuple-based data representation has some disadvantages when compared to the input data representation used in this thesis. We discussed these disadvantages in Section 3.5.3. Since it is very computationally hard to find an optimal

hybrid fragmentation, authors do not compare the obtained results with the optimal fragmentation scheme; rather they compare their solution with an attribute-only and a random fragmentation. The experiments show that the proposed algorithm reduces the execution cost of the queries up to 70%.

Cheng, Wong and Lee use a genetic algorithm-based clustering to fragment data tables both vertically and horizontally [9]. Authors formulate the partitioning problem as the well-known traveling salesman (TCP) problem. For horizontal fragmentation Cheng et al. represent input data as a transaction-predicate matrix. The matrix is then decomposed into submatrices, which determine the resulting set of fragments. It is not easy to tell how exactly the matrix should be decomposed though. Authors propose to formulate this problem as the TSP problem and to solve it using the distance values between transaction and predicate pairs. Based on these values a path for predicates and a path for transactions are calculated using the proposed genetic algorithm-based TSP solution. The paths are cut then by the edges which contribute the most to the costs. Authors do not conduct extensive experiments on their fragmentation algorithm; they just mention that the results obtained by it and the results generated by the algorithm proposed by Zhang and Orlowska are the same. Which allows us to argue that our clustering-based solution might produce better results than those generated by the genetic algorithm-based solution proposed in this paper.

Thenmozhi and Vivekanandan [82] use a genetic algorithm applied to referential horizontal fragmentation in data warehouses. Authors propose two hybrid solutions for the problem: a genetic algorithm combined with hill climbing technique and a genetic algorithm combined with tabu search. Authors claim that the genetic algorithm combined with hill climbing can help overcome the weaknesses of both approaches considered in isolation: hill climbing can easily get stuck in a local optimum, and a genetic algorithm might generate worse solutions with an increasing problem size. So the idea behind the proposed approach is to use a genetic algorithm to find good solutions and hill climbing technique to quickly optimize them. Tabu search method is designed to assure that the recently considered solutions are not re-entered again. Authors use tabu search method to inhibit solutions from being selected more than a specified number of times by moving best solutions obtained in each iteration into the tabu list. Proposed solutions are compared by comparing their execution time, execution cost of the queries on the generated fragmentation schemes and the number of fragments the algorithms generate. Authors state that genetic algorithm combined with with tabu search requires less time than genetic algorithm combined with hill climbing technique while generating even better results. However, the algorithms generate different number of fragments, which depending on the cost function used might be the real reason why the quality of the results differs.

The problem of the dynamic horizontal fragmentation in data warehouses was addressed in [10]. Authors introduce the way of using genetic algorithms both for static and dynamic fragmentation. Chromosome encoding for the genetic algorithm is defined as attribute domain partitioning, i.e. an array of attribute vectors and each element of the vector corresponding to a subdomain of the attribute. To each subdomain a number is

assigned and subdomains with the same identifiers are merged together. Authors aim to make the encoding representation for dynamic fragmentation flexible. They introduce several selection strategies: naive incremental selection that uses `MERGE` operator to adjust the fragmentation scheme, incremental selection based on the genetic algorithm and ameliorated incremental selection based on the genetic algorithm. Although dynamic aspect of the proposed solution is not really relevant for us, we would like to discuss the way authors represent the fragments. The proposed algorithm fetches simple predicates from the queries and for each predicate it divides domain of the attribute into subdomains according to the value used in the predicate. This approach is very similar to the one we use in our RL-based solution, where the size of the observation space must be fixed, and therefore such data representation is a good option to choose. This approach is also suitable for dynamic fragmentation, since the size of the chromosome encoding remains the same when new queries arrive. However, we would like to point out that in general, because of the reasons mentioned in Section 3.5.3, for the static fragmentation problem it might be more efficient to use selected in this thesis queries-based data representation for chromosome encoding in genetic algorithms.

Another metaheuristic that is often applied to optimization problems is PSO (Particle swarm optimization). PSO algorithms start with a set of random solutions (particles) and move these particles in a search space towards an optimal solution.

Database fragmentation can also be performed using PSO algorithms. For instance, in [83] authors formulate fragmentation problem as an optimization problem and use a PSO algorithm to find an optimal fragmentation scheme for data warehouses. In data warehouse environment an important requirement for a fragmentation scheme is a fixed number of fragments. Proposed PSO-based fragmentation algorithm controls the number of generated fragments while trying to minimize an objective function. The algorithm places particles randomly in the search space and assign them an initial direction. The particles then move towards or away from each other according to the specified rule of local behaviour. Sets of particles that are moving together form fragments. Authors test the results using Oracle database and compare them with the results generated using range partitioning technique and classical horizontal fragmentation algorithms (affinity- and predicate-based algorithms). Additionally, authors compare their results with a non-fragmented database table. Although the experiments do not show much improvement over range partitioning technique, proposed by the authors PSO-based fragmentation algorithm outperforms affinity- and predicate-based algorithms. However, it is worthwhile to mention that unlike our affinity-based algorithm adaptation their predicate- and affinity-based algorithms do not limit the number of fragments (there are 150 fragments generated by predicate-based algorithm, 115 - by affinity-based algorithm and only 8 fragments generated by the PSO-based solution). That is why these experiments are not quite representative.

PSO and genetic algorithms are similar to each other [84] in the sense that they are population-based; the members of the population interact with each other and share information in order to find an optimal solution for the problem. The process is guided by a set of probabilistic and deterministic rules. Both of them can be applied to

the fragmentation task. However, in this thesis we wanted to provide a simple and transparent alternative approach for RL-based horizontal fragmentation algorithm, that is why we decided to use one of the classical clustering algorithms with limited number of configuration parameters.

6.4 Summary

In this chapter we discuss and analyse papers that solve similar problems of physical database design using Machine Learning. First, we review some papers that use clustering-based approaches for solving the issues related to horizontal fragmentation. Next, we show how Reinforcement Learning is used for different physical design problems. Last, we discuss the papers that use other optimization algorithms for the database fragmentation task.

7. Conclusion and Future work

In this chapter we summarise our work, discuss possible threats to its validity and suggest directions for future research.

7.1 Work summary

In this thesis we evaluate the applicability of ML approaches to the task of primary horizontal fragmentation. We focus on two specific ML techniques: Unsupervised Learning (Clustering) and Deep Reinforcement Learning.

In this thesis we proposed two **clustering-specific** research questions. We answer the first one by designing a clustering-based horizontal fragmentation approach in Section 3.5 and comparing it to the classical horizontal fragmentation algorithm proposed by Zhang and Orlowska in Chapter 5. The second question is answered in Section 5.2 by testing the impact of the following configuration parameters of our clustering-based fragmentation solution: cost model usage, linkage criterion and similarity measure.

We answer the **RL-specific** research questions by designing the Deep-RL-based horizontal fragmentation solution in Section 3.6 and evaluating the impact of the agent choice and update horizon hyper-parameter as well as generalization of the workload in Section 5.3.

There are multiple comparison criteria, which can be used to evaluate the quality of a horizontal fragmentation solution. Based on the research we have done to answer the questions from Section 1.2 and the evaluation results from Chapter 5 we provide multifaceted comparative analysis of the features of our approaches in Table 7.1.

criterion	classical affinity-based solution	clustering-based solution	RL-based solution

strategy	top-down	bottom-up	top-down
data representation	predicate-dependent	data-dependent	data- and predicate-dependent
cost model usage	optional	optional	required
fragmentation correctness rules	does not guarantee disjointness of the generated fragmentation scheme	guarantees completeness, reconstructability and disjointness of the generated fragmentation scheme	guarantees completeness, reconstructability and disjointness of the generated fragmentation scheme
quality of the results	baseline	generally better results	results vary in quality, being, however, better than the baseline
speed	is in general faster than the clustering-based solution (except the clustering-based solution with the Penalty-based method)	the Penalty-based method allows the clustering-based algorithm to generate results faster than the classical fragmentation algorithm	is comparable to clustering during training; fully trained agent generates a fragmentation scheme almost immediately (not using the cost model)
scalability	with the increase in the number of unique simple predicates, the time complexity of the classical fragmentation solution increases rapidly	with the increase in the number of queries, the time complexity of the classical fragmentation solution increases	with the increase in the number of columns, fragments or queries, the time complexity of the classical fragmentation solution increases

limitations	it can difficult to split the real-world user queries into simple predicates; similarity between predicates is defined using their usage in the queries, rather than the amount of common data they access	since the algorithm is data-dependent if the data in the table was changed, generated fragmentation scheme might not be optimal anymore	only range predicates in queries and fragments; fixed number of maximum fragments

Table 7.1: Comparative analysis of the solutions

Generally, the clustering-based algorithm outperforms other solutions in regard to the quality of the generated fragmentation schemes. However, fully trained Deep-RL solution has a benefit of generating the results much faster, which can prove useful for rapidly changing workloads. Both ML-based approaches tend to outperform the baseline in both regards.

7.2 Threats to validity

- **Different input data representation:** our ML approaches use different input data representation and fragmentation strategies; therefore, some differences in the evaluation results might be caused by this reason.

- **Imperfect cost model:** in our research we used a cost model to speed up the fragmentation process. We tried to capture the core optimization concepts used in modern DBMSs. However, there a real database system might behave differently on the generated fragmentation schemes.

- **Number of workloads:** due to time constraints we test the solutions on a limited number of workloads. Running more tests would result in higher accuracy of the experiments. This is especially important for Deep-RL based solution, since it requires a large number of test data to fully generalize.

- **Testbench limitations:** The solutions are tested on single table from TPC-H benchmark. To allow for more robust performance comparison, the solutions might need more diverse set of input data.

- **Deep-RL hyper-parameters:** we test the impact of only small subset of hyper-parameters on convergence of Deep-RL-based solution. Some of the untested parameters might also have influence on the training process.
- **Deep-RL training time:** some of the artifacts in the results of the RL-based solutions might have been caused by limited time we had for the training. Better quality of the results might be achieved by allowing the agents to train for a longer time.
- **Baseline algorithms:** due to the time constraints we could not adapt more than one classical horizontal fragmentation algorithm to the assumptions made in this thesis. There might be more suitable baselines for our work to compare against.

7.3 Future work

Our thesis provides insights into the applicability of two very different ML approaches to the task of primary horizontal fragmentation. To be able to compare them, we made a number of assumptions. However, some of these assumptions can be eliminated to allow for more universally applicable solutions. Moreover, the analysis of related works allowed us to formulate further directions for future work.

Improvements that can be made to our solutions in general are:

- **Scalability.** Neither of the proposed solutions works well on the large workloads and to improve the scalability one can use various methods for reducing the search space (e.g. pre-processing of the queries as suggested in [76])
- **Cost model accuracy.** As mentioned above, in the future we would like to use more accurate tools for evaluating fragmentation scheme performance. There exist tools for low-level simulation of a fragmented database (HypoPG for PostgreSQL), which, unfortunately, do not provide the functionality we need yet.
- **Distributed environment.** In this thesis we limit our problem to a horizontal fragmentation in non-distributed environment. However, many databases are distributed nowadays and the tasks of allocation and replication are often considered alongside with the fragmentation task. Therefore, in the future work we would like to adapt our solutions to the distributed environment.
- **Query statistics.** We believe that including the frequencies, with which the queries are executed, into the data representation and the fragmentation process might provide more meaningful results.
- **Other ML techniques.** As discussed in the related work chapter, there are multiple ML-based optimization algorithms that can be used for solving physical design tasks. In this thesis we only considered two of them, but comparison with other optimization algorithms might be interesting for the further research.

The future work directions specific to the clustering-based solution are:

- **Evolving set of queries.** Current implementation of the clustering-based fragmentation solution is not able to adapt when new queries arrive. Since rerunning of the clustering algorithm in this case might be time-consuming, approaches for dynamic fragmentation similar to those used in [72, 73] can be applied to make the clustering-based solution more flexible.
- **Other clustering algorithms.** Due to the reasons mentioned in Section 3.5.2, we use a hierarchical clustering algorithm in this thesis. However, in the future some of the assumptions we made in the work might not be relevant anymore, and other clustering algorithms such as partitioning-based or grid-based algorithms might be considered.

Possible improvements specific to our Deep-RL-based approach consist of:

- **Predicate type generalization.** The assumptions made in the beginning of this thesis limit the type of query and fragment predicates, which can be used with Deep-RL solution. We would like to lift this limitation by considering other data representation and environment designs and evaluate their performance.
- **Dynamic number of fragments.** Another hard limitation of Deep-RL solution is a fixed maximum number of fragments. The future implementation of this approach might consider the problem formulation, where maximum number of fragments is an input parameter instead of a hyper-parameter (i.e. it can be changed without re-training the Deep-RL model).
- **Deep-RL agents.** The evaluation chapter of this thesis only considers tests on two agents provided by Dopamine framework: Rainbow and Implicit Quantile. We would like to expand this list and conduct experiments on more agents in the future.

Bibliography

[1] Y. Zhang and M. E. Orlowska, "On fragmentation approaches for distributed database design," Information Sciences-Applications, vol. 1, no. 3, pp. 117–132, 1994. (cited on Page iii, 2, 14, 16, 59, 63, 64, 85, 87, and 99)

[2] M. T. Özsu and P. Valduriez, Principles of distributed database systems. Springer Science & Business Media, 2011. (cited on Page iii, 2, 13, 14, 15, and 16)

[3] S. Chaudhuri and V. Narasayya, "Self-tuning database systems: A decade of progress," in Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, pp. 3–14, VLDB Endowment, 2007. (cited on Page 1)

[4] R. Borovica, I. Alagiannis, and A. Ailamaki, "Automated physical designers: what you see is (not) what you get," in Proceedings of the Fifth International Workshop on Testing Database Systems, p. 9, ACM, 2012. (cited on Page 2)

[5] L. Bellatreche, K. Boukhalfa, P. Richard, and K. Y. Woameno, "Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms," International Journal of Data Warehousing and Mining (IJDWM), vol. 5, no. 4, pp. 1–23, 2009. (cited on Page 2 and 14)

[6] R. Wirth and J. Hipp, "Crisp-dm: Towards a standard process model for data mining," in Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining, pp. 29–39, Citeseer, 2000. (cited on Page 4)

[7] D. Nashat and A. A. Amer, "A comprehensive taxonomy of fragmentation and allocation techniques in distributed database design," ACM Computing Surveys (CSUR), vol. 51, no. 1, p. 12, 2018. (cited on Page 9)

[8] S. Ceri, M. Negri, and G. Pelagatti, "Horizontal data partitioning in database design," in Proceedings of the 1982 ACM SIGMOD international conference on Management of data, pp. 128–136, ACM, 1982. (cited on Page 10 and 14)

[9] C.-H. Cheng, W.-K. Lee, and K.-F. Wong, "A genetic algorithm-based clustering approach for database partitioning," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 32, no. 3, pp. 215–230, 2002. (cited on Page 13 and 110)

[10] R. Bouchakri, L. Bellatreche, Z. Faget, and S. Breß, "A coding template for handling static and incremental horizontal partitioning in data warehouses," Journal of Decision Systems, vol. 23, no. 4, pp. 481–498, 2014. (cited on Page 13 and 110)

[11] W. T. McCormick Jr, P. J. Schweitzer, and T. W. White, "Problem decomposition and data reorganization by a clustering technique," Operations Research, vol. 20, no. 5, pp. 993–1009, 1972. (cited on Page 16)

[12] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," ACM Transactions on Database Systems (TODS), vol. 9, no. 4, pp. 680–710, 1984. (cited on Page 16, 17, and 18)

[13] S. Navathe, K. Karlapalem, and M. Ra, "A mixed fragmentation methodology for initial distributed database design," Journal of Computer and Software Engineering, vol. 3, no. 4, pp. 395–426, 1995. (cited on Page 20)

[14] S. B. Navathe and M. Ra, "Vertical partitioning for database design: a graphical algorithm," in ACM Sigmod Record, vol. 18, pp. 440–450, ACM, 1989. (cited on Page 20)

[15] D.-G. Shin and K. B. Irani, "Fragmenting relations horizontally using a knowledge-based approach," IEEE Transactions on Software Engineering, no. 9, pp. 872–883, 1991. (cited on Page 21)

[16] N. Khalil, D. Eid, and M. Khair, "Availability and reliability issues in distributed databases using optimal horizontal fragmentation," in International Conference on Database and Expert Systems Applications, pp. 771–780, Springer, 1999. (cited on Page 21)

[17] S. I. Khan and A. Hoque, "A new technique for database fragmentation in distributed systems," International Journal of Computer Applications, vol. 5, no. 9, pp. 20–24, 2010. (cited on Page 21)

[18] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker, "E-store: Fine-grained elastic partitioning for distributed transaction processing systems," Proceedings of the VLDB Endowment, vol. 8, no. 3, pp. 245–256, 2014. (cited on Page 21)

[19] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 48–57, 2010. (cited on Page 21)

[20] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, "Distributed data placement via graph partitioning," arXiv preprint arXiv:1312.0285, 2013. (cited on Page 22)

[21] M. Serafini, R. Taft, A. J. Elmore, A. Pavlo, A. Aboulnaga, and M. Stonebraker, "Clay: fine-grained adaptive partitioning for general database schemas," Proceedings of the VLDB Endowment, vol. 10, no. 4, pp. 445–456, 2016. (cited on Page 22)

[22] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems," in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 61–72, ACM, 2012. (cited on Page 22)

[23] C. Bishop, Pattern recognition and machine learning. New York: Springer, 2006. (cited on Page 22)

[24] S. Marsland, Machine learning: an algorithmic perspective. Chapman and Hall/CRC, 2014. (cited on Page 23 and 24)

[25] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," IEEE transactions on emerging topics in computing, vol. 2, no. 3, pp. 267–279, 2014. (cited on Page 26, 28, and 29)

[26] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, Introduction to Data Mining (2Nd Edition). Pearson, 2nd ed., 2018. (cited on Page 26 and 27)

[27] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," ACM Transactions on Database Systems (TODS), vol. 42, no. 3, p. 19, 2017. (cited on Page 28)

[28] I. MR and D. MOHAN, "A survey of grid based clustering algorithms," International Journal of Engineering Science and Technology, vol. 2, 08 2010. (cited on Page 29)

[29] E. Schubert, "Knowledge discovery in databases. part iii. clustering." (cited on Page 30, 31, 32, 34, 61, and 86)

[30] Y. Z. Mohith Manjunath, "Exploring patterns in big data using clustereng: A clustering engine for genomics." (cited on Page 30)

[31] N. S. Software, "Hierarchical clustering / dendrograms." (cited on Page 32)

[32] A. S. Darabant and A. Campan, "Semi-supervised learning techniques: k-means clustering in oodb fragmentation," in Second IEEE International Conference on Computational Cybernetics, 2004. ICCC 2004., pp. 333–338, IEEE, 2004. (cited on Page 35 and 106)

[33] A. S. Darabant and A. Campan, "Ai clustering techniques: a new approach to object oriented database fragmentation," in Proceedings of the 8th IEEE International Conference on Intelligent Engineering Systems, Cluj Napoca, pp. 73–78, 2004. (cited on Page 35 and 36)

[34] A. S. Darabant and A. Câmpan, "Advanced object database design techniques," Carpathian Journal of Mathematics, pp. 21–30, 2004. (cited on Page 35)

[35] A. S. Darabant and A. Gog, "Hierarchical clustering in large object datasets-a study on complexity, quality and scalability.," Studia Universitatis Babes-Bolyai, Informatica, no. 2, 2009. (cited on Page 35)

[36] F. Baião and M. Mattoso, "A mixed fragmentation algorithm for distributed object oriented databases," in Proc Int'l Conf Computing and Information (ICCI'98), Winnipeg, pp. 141–148, 1998. (cited on Page 35)

[37] L. Bellatreche, K. Karlapalem, and A. Simonet, "Horizontal class partitioning in object-oriented databases," in International Conference on Database and Expert Systems Applications, pp. 58–67, Springer, 1997. (cited on Page 35)

[38] A. S. Darabant, A. Campan, and O. Cret, "Hierarchical clustering in object oriented data models with complex class relationships," in Proceedings of the 8th IEEE International Conference on Intelligent Engineering Systems, Cluj Napoca, pp. 307–312, 2004. (cited on Page 36)

[39] A. S. Darabant, "A new approach in fragmentation of distributed object oriented databases using clustering techniques," Studia Univ. babes, L (2), 2005. (cited on Page 36)

[40] A. Darabant, A. Câmpan, G. Moldovan, and H. Grebla, "Ai clustering techniques: a new approach in horizontal fragmentation of classes with complex attributes and methods in object oriented databases," in the Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics-ICTAMI, pp. 109–128, 2004. (cited on Page 36)

[41] A. S. Darabant, H. Todoran, O. Cret, and G. Chis, "The similarity measures and their impact on oodb fragmentation using hierarchical clustering algorithms.," WSEAS Transactions on Computers, vol. 5, no. 9, pp. 1803–1810, 2006. (cited on Page 36 and 68)

[42] A. S. Darabant, H. Todoran, O. Cret, and G. Chis, "A comparative study on the influence of similarity measures in hierarchical clustering in complex distributed object-oriented databases," in Proceedings of the 10th WSEAS international conference on Computers, pp. 235–240, World Scientific and Engineering Academy and Society (WSEAS), 2006. (cited on Page 36)

[43] A. S. Darabant, A. Campan, and O. Cret, "Using fuzzy clustering for advanced oodb horizontal fragmentation with fine-grained replication.," in Databases and Applications, pp. 116–121, Citeseer, 2005. (cited on Page 36)

[44] A. S. Darabant and L. Darabant, "Clustering methods in data fragmentation," Rom. Journ. of Information Science and Technology, vol. 14, no. 1, pp. 81–97, 2011. (cited on Page 36)

[45] C. Ezeife and K. Barker, "Horizontal class fragmentation for advanced-object modes in a distributed object-based system"," in the Proceedings of the 9th International Symposium on Computer and Information Sciences, pp. 25–32, 1994. (cited on Page 36)

[46] A. Cuzzocrea, J. Darmont, and H. Mahboubi, "Fragmenting very large xml data warehouses via k-means clustering algorithm," International Journal of Business Intelligence and Data Mining, vol. 4, no. 3/4, p. 301, 2009. (cited on Page 39, 59, and 64)

[47] H. Mahboubi and J. Darmont, "Data mining-based fragmentation of xml data warehouses," 2008. (cited on Page 39)

[48] L. Rodríguez-Mazahua, G. Alor-Hernández, M. A. Abud-Figueroa, and S. G. Peláez-Camarena, "Horizontal partitioning of multimedia databases using hierarchical agglomerative clustering," in MICAI, 2014. (cited on Page 39 and 64)

[49] S. Harikumar and R. Ramachandran, "Hybridized fragmentation of very large databases using clustering," in 2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES), pp. 1–5, IEEE, 2015. (cited on Page 39)

[50] V. N. Luong, H. H. C. Nguyen, and V. S. Le, "An improvement on fragmentation in distribution database design based on knowledge-oriented clustering techniques," ArXiv, vol. abs/1505.01535, 2015. (cited on Page 39 and 63)

[51] V. N. Luong, H. H. C. Nguyen, and V. S. Le, "An improvement on fragmentation in distribution database design based on clustering techniques," 2015. (cited on Page 39 and 63)

[52] S. Hirano and S. Tsumoto, "A knowledge-oriented clustering technique based on rough sets," in 25th Annual International Computer Software and Applications Conference. COMPSAC 2001, pp. 632–637, IEEE, 2001. (cited on Page 39)

[53] I. Hamdi, E. Bouazizi, S. Alshomrani, and J. Feki, "2lpa-rtdw: A two-level data partitioning approach for real-time data warehouse," 2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS), pp. 632–638, 2015. (cited on Page 40 and 64)

[54] T. T. Nguyen, B. Van Doan, C. N. Truong, and T. T. T. Tran, "Clustering and query optimization in fuzzy object-oriented database," International Journal of Natural Computing Research (IJNCR), vol. 8, no. 1, pp. 1–17, 2019. (cited on Page 40)

[55] L. Sun, Skipping-oriented Data Design for Large-Scale Analytics. PhD thesis, EECS Department, University of California, Berkeley, Dec 2017. (cited on Page 40)

[56] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning. Cambridge, MA, USA: MIT Press, 1st ed., 1998. (cited on Page 43, 44, 45, 47, and 48)

[57] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992. (cited on Page 47)

[58] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," Neurocomputing, vol. 234, pp. 11–26, 2017. (cited on Page 49)

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. (cited on Page 50 and 51)

[60] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. (cited on Page 51)

[61] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. (cited on Page 51)

[62] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017. (cited on Page 52)

[63] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," 2018. (cited on Page 52 and 53)

[64] L. Sun, M. J. Franklin, J. Wang, and E. Wu, "Skipping-oriented partitioning for columnar layouts," Proceedings of the VLDB Endowment, vol. 10, no. 4, pp. 421–432, 2016. (cited on Page 57)

[65] G. Huang, D. Wang, and J. Ren, "Grid and density based clustering algorithm with relative entropy," Advances in Information Sciences and Service Sciences, vol. 5, no. 2, p. 36, 2013. (cited on Page 61)

[66] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A research framework for deep reinforcement learning," 2018. (cited on Page 70)

[67] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. (cited on Page 70)

[68] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," Journal of Artificial Intelligence Research, vol. 47, p. 253–279, Jun 2013. (cited on Page 71)

[69] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," 2018. (cited on Page 71 and 72)

[70] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," 2018. (cited on Page 76)

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. (cited on Page 84)

[72] A. Campan, A. S. Darabant, and G. Serban, "Clustering techniques for adaptive horizontal fragmentation in object oriented databases," in Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics ICTAMI, pp. 263–274, 2005. (cited on Page 105 and 117)

[73] A. S. Darabant, A. Câmpan, H. Todoran, and G. Serban, "Incremental horizontal fragmentation: A new approach in the design of distributed object oriented databases," ICCCC 2006, p. 170, 2006. (cited on Page 105 and 117)

[74] G. Şerban and A. Câmpan, "Hierarchical adaptive clustering," Informatica, vol. 19, no. 1, pp. 101–112, 2008. (cited on Page 106)

[75] R. Ramachandran, D. P. Nair, and J. Jasmi, "A horizontal fragmentation method based on data semantics," in 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1–5, IEEE, 2016. (cited on Page 106)

[76] G. Amina and K. Boukhalfa, "Very large workloads based approach to efficiently partition data warehouses," in Modeling Approaches and Algorithms for Advanced Computer Applications, pp. 285–294, Springer, 2013. (cited on Page 106, 107, and 116)

[77] G. Campero Durand, M. Pinnecke, R. Piriyev, M. Mohsen, D. Broneske, G. Saake, M. Sekeran, F. Rodriguez, and L. Balami, "Gridformation: Towards self-driven online data partitioning using reinforcement learning," pp. 1–7, 06 2018. (cited on Page 107)

[78] G. Campero Durand, R. Piriyev, M. Pinnecke, D. Broneske, B. Gurumurthy, and G. Saake, Automated Vertical Partitioning with Deep Reinforcement Learning, pp. 126–134. 09 2019. (cited on Page 107)

[79] B. Hilprecht, C. Binnig, and U. Röhm, "Towards learning a partitioning advisor with deep reinforcement learning," in Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM '19, (New York, NY, USA), pp. 6:1–6:4, ACM, 2019. (cited on Page 108)

[80] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management - aiDM'18, 2018. (cited on Page 108 and 109)

[81] N. Gorla, V. Ng, and D. M. Law, "Improving database performance with a mixed fragmentation design," Journal of intelligent information systems, vol. 39, no. 3, pp. 559–576, 2012. (cited on Page 109)

[82] M. Thenmozhi and K. Vivekanandan, "A comparative analysis of fragmentation selection algorithms for data warehouse partitioning," in 2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014), pp. 1–5, IEEE, 2014. (cited on Page 110)

[83] H. Derrar, M. Ahmed-Nacer, and O. Boussaid, "Particle swarm optimisation for data warehouse logical design," International Journal of Bio-Inspired Computation, vol. 4, no. 4, pp. 249–257, 2012. (cited on Page 111)

[84] R. Hassan, B. Cohanim, O. De Weck, and G. Venter, "A comparison of particle swarm optimization and the genetic algorithm," in 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference, p. 1897, 2005. (cited on Page 111)