Presentation of paper "Learning Multi-dimensional Indexes"

Christina Borovilou
ds1200008@di.uoa.gr
University of Athens, Department of Informatics and Telecommunications
Athens, Greece

Figure 1: Indexing before computers' era

ABSTRACT

As computer science becomes more sophisticated, hardware more powerful, data more massive and users more demanding, the need for strong and fast search is loud. We are in an era that we should exceed the capabilities of our tools, exploit logical relations among data in order to save unnecessary computational effort and to build strong systems to serve people's needs in every aspect. Searching for data is a fundamental practice that needs improvement in parallel with the technological evolution.

CCS CONCEPTS

• Theory of computation → Data structures and algorithms for data management; **• Information systems** → *Database query processing*; Data analytics.

KEYWORDS

Datasets, learning index, Multi-dimensional Index, MIT, Machine Learning

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DMAW 2021, September 24, 2021, Athens, GR
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Christina Borovilou. 2021. Presentation of paper "Learning Multi-dimensional Indexes". In *Proceedings of Data Management and Applications Workshop (DMAW 2021).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This work is a part of an educational project under the course "Database Systems" of Master Program DSIT of NKUA. The information discussed below is triggered by a recent paper of Vikram Nathan, Jialin Ding, Mohammad Alizadeh and Tim Kraska concerning Learning Multi-dimensional Indexes [?]. In this presentation, we will give an overview of the things the paper is proposing, procedures that reevaluates and ideas that are generated. We will demonstrate some ideas discussed at the paper that we implemented in the scope of the course project. We will showcase the ideas behind, how they worked and we will point out observations as well as pain points of the attempt.

2 PAPER OVERVIEW

The team in this paper after presenting the current "tools in Indexing industry", introduces a new index, named "Flood". Starting from the description of the new index, "Learning, Multi-dimensional Index" we distinguish two equal complicated characteristics of the index: it is multi-dimensional and also seems to commit training in order to learn and adapt. Thus, we will examine indexing from two different perspectives:

Unpublished working draft. Not for distribution.

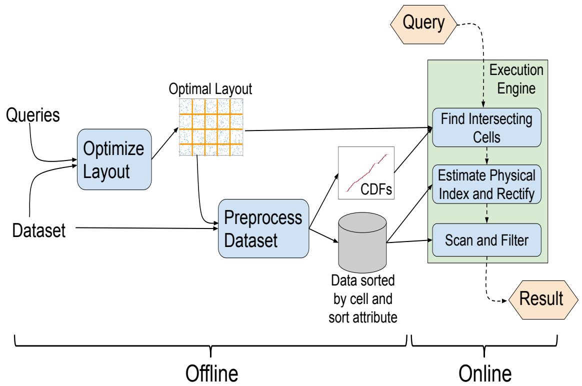


Figure 2: Flood's Architecture

Learning Index: They introduce the learning index to be the result of 2 important parameters: Learning from user's queries (frequency, ranges and correlations) as well as learning from data itself, in the sense that instead of adapting our Queries and optimizers to computer architecture, we can adapt our data architecture to the characteristics of our Database. As they notice though experiments to databases with different characteristics (Skewness, Size, Dimensionality, correlated attributes) query time parameters vary lot and do not follow a pattern that can be modeled analytically. And when math do not serve the problem, Machine Learning is called to find the parameters that make the system faster. To do so, Flood's team supports that query time is mainly the combination of 3 parameters according to its architecture:

- Projection: Identify the cells in the grid layout (we will explain later on) that mach the filter of `SELECT` statement
- Refinement: responsible to decide which attributes should be ordered and in which sequence
- Scan: time depending on users's number of filters, data distribution

These parameters do not contribute equally to the total time. Floods team uses ML algorithms to calculate these hidden weights. Flood trains a random forest regression model to predict them based on the output statistics. At this point they reach an interesting finding: The weight models are accurate across different datasets and query workloads! Independently of new data insertion or changes over query distribution, Flood will only reevaluate the existing models, will not restart the leaning process from the start!

Multi-Dimensional Index: For sure we cannot order data in more than one dimensions; but we can find heuristic ways to simulate it. We all know the value of binary search in query response. The core of that heuristic technique is that it imposes relational logic among data and exploits it when it is time. So if the one who "serves" data is the one who is storing data (meaning Database System) many techniques can be developed to the favor of easier access. That is what the Flood team introduces: a "geometrical" storage trick that help queries to feel more lucky even when index is on the filter.

SELECT SUM(R.X)
FROM MyTable
WHERE (a ≤ R.Y ≤ b) AND (c ≤ R.Z ≤ d)

Flood receives as input a filter predicate consisting of ranges over one or more attributes, joined by ANDs. The intersection of these ranges defines a hyper-rectangle, and Flood's goal is to find and process exactly the points within this hyperrectangle (e.g., by aggregating them). This feature we will examine in the Implementation Section.

Note that equality predicates of the form R.Z R.Z == f can be rewritten as f ≤ R.Z ≤ fx. We should mention that Floods work is focused on "ANDs" operations on `WHERE` clause. Equality predicates of the form R.Z == f can be rewritten as f ≤ R.Z ≤ f. Disjunctions (i.e. OR clauses) can be decomposed into multiple queries over disjoint attribute ranges.

3 IMPLEMENTATION - OPTIMIZING THE GRID

Since the paper did not provide in public the code, my implementation was not to expand the work that was done but try to reproduce some part of it. In the following steps i will demonstrate the creation a new layout, adapted to specific attributes (Sequence of the attributes DOES matter). Sequence of attributes for the creation of layout in the scope of paper is given by ML procedures in the first part. In the code of the implementation is arbitrary.

3.1 Grid Layout Density

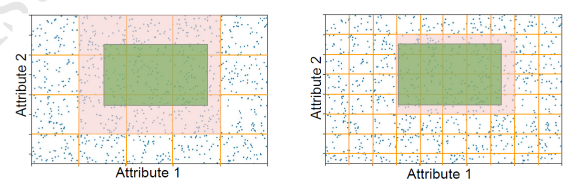


Figure 3: Scan Overhead

Layout is visualized above and is the grid that 2 (or more in the N-dimensional space) attributes scatter their data. The 2 representations above show off the query optimization dilemma: Split the space with dense grid or to a more sparse one? The first will achieve low scan overhead (it is the red area of the grid and corresponds to scanned points that didn't meet the selection criteria). but that would result to many visits (time rises up) in each cell and will counterbalance the profit of low scan overhead. The minimal loss will be given if we ask data. That process is in the ML process we mentioned in Section 2.

3.2 Data Distribution Skewness

Flood is also concerned about equally distributing data points along the layout. Using pandas in python we process our data creating equally distributed data in the grid and not a homogeneously in space (that will not offer any added value)

That feature will be used in the next section.

Unpublished working draft. Not for distribution.

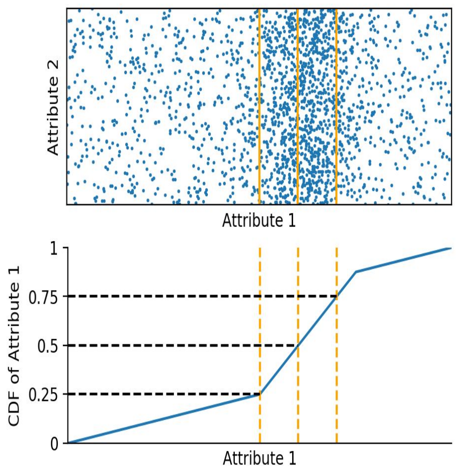


Figure 4: Skewed Data, equally distributed

3.3 Grouping and ordering Compilation

Next step is to build a multi-dimensional index.

The main idea is described below: Take d-1 attributes to create grouping in your data. Let the last to be the sorting dimension. According to homogeneously flatten data in each dimension, make groups. Lets imagine the example of having dimensions in our dataset (60 rows): (Student Number, grade, Grad.Year) with ranges [100-160],[5-10],[2010-2021]. We split them all in to 3 sub ranges. From our statistics, we select Student number to be the first group with sub range [100-120]. Then sub ranges [5-6],[6-7],[8-10] of grad.year should order inside that group in order. In this case we order just by group not bu value. And the last attribute will be sorted in each list separately - not in the whole dataset. Illustrations of the manipulation of the data are shown at figure 5. In this case the query:

SELECT price
WHERE 2018 > grad. Year > 2015
AND 125 >AM >105
AND grade > 7

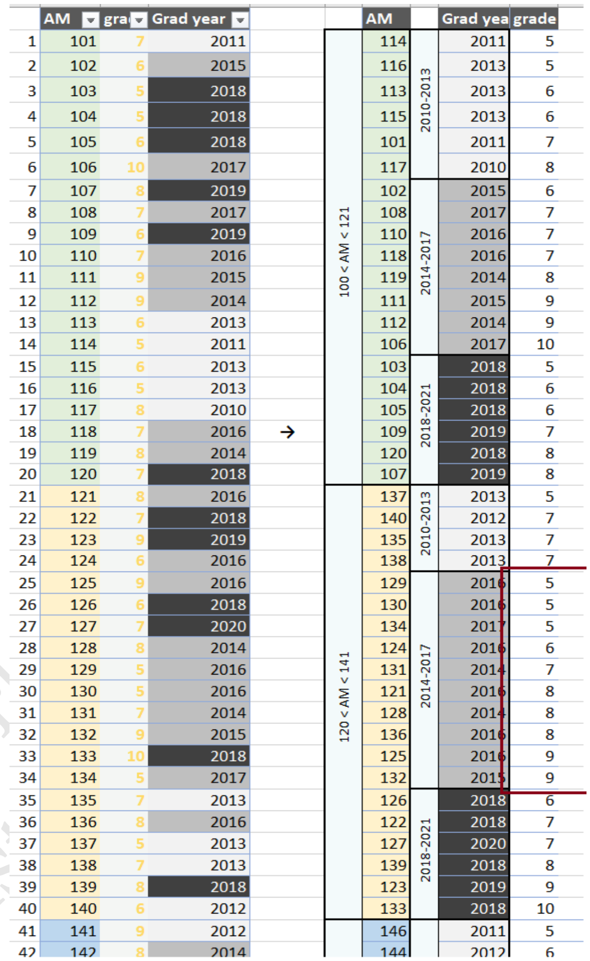
will search with binary search over prices (sorted attribute). That method literally can be thought as a generalization of binary search or B+ tree search with more tolerance in the values of the tree in order more dimensions to fit the relational logic among data and to achieve faster results.

4 PYTHON NOTEBOOK AND TESTS

In the colab notebook you would be ask to mount your file and give the data path. Also the user can select how dense or spare will be the layout in order to compare the two methodologies. At the github folder we have provided some dataset that where used as test dataset

5 CONCLUSIONS

There are a lot more to discover in the new era of Machine Learning. Computers exceed human limitation and help them overcome obstacles. The results of Flood team reach above 100percent in some cases and id someone see the performance related to "typical: indexes, the difference is terrifying if we can think of what is next. Problems adaptability and universality are about ti be developed in the upcoming years



	AM	gra	Grad year			AM		Grad yea	grade
1	101	7	2011		100 < AM < 121	114	2010-2013	2011	5
2	102	6	2015			116		2013	5
3	103	5	2018			113		2013	6
4	104	5	2018			115		2013	6
5	105	6	2018			101		2011	7
6	106	10	2017			117		2010	8
7	107	8	2019			102	2014-2017	2015	6
8	108	7	2017			108		2017	7
9	109	6	2019			110		2016	7
10	110	7	2016			118		2016	7
11	111	9	2015			119		2014	8
12	112	9	2014			111		2015	9
13	113	6	2013			112		2014	9
14	114	5	2011			106		2017	10
15	115	6	2013			103	2018-2021	2018	5
16	116	5	2013			104		2018	6
17	117	8	2010			105		2018	6
18	118	7	2016	→		109		2019	7
19	119	8	2014			120		2018	8
20	120	7	2018			107		2019	8
21	121	8	2016		120 < AM < 141	137	2010-2013	2013	5
22	122	7	2018			140		2012	7
23	123	9	2019			135		2013	7
24	124	6	2016			138		2013	7
25	125	9	2016			129	2014-2017	2016	5
26	126	6	2018			130		2016	5
27	127	7	2020			134		2017	5
28	128	8	2014			124		2016	6
29	129	5	2016			131		2014	7
30	130	5	2016			121		2016	8
31	131	7	2014			128		2014	8
32	132	9	2015			136		2016	8
33	133	10	2018			125		2016	9
34	134	5	2017			132		2015	9
35	135	7	2013			126	2018-2021	2018	6
36	136	8	2016			122		2018	7
37	137	5	2013			127		2020	7
38	138	7	2013			139		2018	8
39	139	8	2018			123		2019	9
40	140	6	2012			133		2018	10
41	141	9	2012			146		2011	5
42	142	8	2014			144		2012	6

Figure 5: Grouping, Subgrouping and Ordering

6 ONLINE RESOURCES

Random Forest Algorithm
Secondary Indexes Python
B Tree
Python with Pandas
Get Skewness
pandas Indexing
The Case for Learned Index Structures

Unpublished working draft. Not for distribution.