University of Lethbridge

DBMS Project Proposal

July 17, 2024

MohammadReza Ardestani
Master's student

Prof. Wendy Osborn
CPSC5660 Instructor

1 Summary of Components

I chose PGM-index paper (Ferragina and Vinciguerra, 2020) to focus on. This recently published paper achieves state-of-the-art performance by incorporating computational geometry concepts.

1.1 Introduction

There are different data structures for indexing which are as follows:

1. Hash-based
2. Tree-based
3. Bitmap-based
4. Trie-based
5. **Learned-indices based**
   (a) ML Based
   (b) **Non-ML based**

An increasing number of learned-based papers is published recently, yet many of them fall into the ML-based category which has its problems. PGM-index paper, unlike others, uses a novel non-ML-based approach detailed below.

1.2 PGM-index details

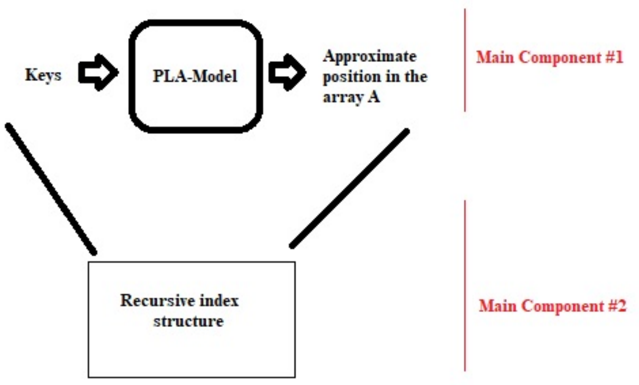
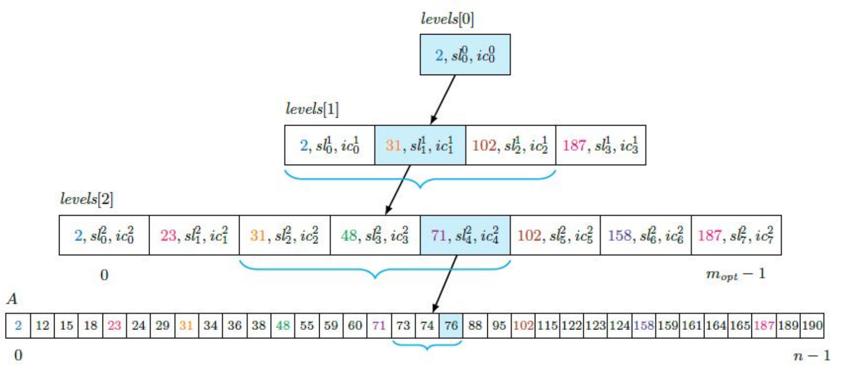

Keys
PLA-Model
Approximate position in the array A
Main Component #1
Recursive index structure
Main Component #2

Figure 1: Main components (photo from myself)

As depicted in Figure1, there are two main components in the PGM model; Peace-wise Linear Approximation and a recursive index structure above it. The first component's implementation boils down to finding the optimal PLA. In the DBMS literature, many approaches have been tried. Dynamic programming, for example, has been used to find the best PLA with $O(n^3) time$ which is not so efficient. The authors, allegedly for the first time, found that a similar problem has been addressed in other areas by using a Rectangle Enclosing the Convex Hull which takes $O(n) time$ time and space. The details of this method or its pseudo-code are not mentioned but referred to [28, 9, 11, 12, 39] in the 2.1 section of the paper for interested readers or anyone who wants to implement the model, like me! The second component deals with creating a tree-based structure. Once we build the PLA for one level, we use the starting points of each segment to build a higher-level segmentation by applying the PLA model until we have only got one segment.



```
BUILD-PGM-INDEX($A, n, \varepsilon$)
levels = an empty dynamic array
$i = 0$; keys = $A$
repeat
    $M$ = BUILD-PLA-MODEL(keys, $\varepsilon$)
    levels[$i$] = $M$; $i = i + 1$
    $m$ = SIZE($M$)
    keys = [$M[0]$.key, ..., $M[m-1]$.key]
until $m = 1$
return levels in reverse order

QUERY($A, n, \varepsilon$, levels, $k$)
pos = $f_r(k)$, where $r$ = levels[0][0]
for $i = 1$ to SIZE(levels) $- 1$
    lo = $\max\{pos - \varepsilon, 0\}$
    hi = $\min\{pos + \varepsilon$, SIZE(levels[$i$]) $- 1\}$
    $s$ = the rightmost segment $s'$ in
        levels[$i$][lo, hi] such that $s'$.key $\le k$
    $t$ = the segment at the right of $s$
    pos = $\lfloor \min\{f_s(k), f_t(t.key)\} \rfloor$
lo = $\max\{pos - \varepsilon, 0\}$
hi = $\min\{pos + \varepsilon, n - 1\}$
return search for $k$ in $A$[lo, hi]
```

Figure 2: Searching for 76 key value, as an example, in the PGM-model built upon the leaves keys. On the right side, a high-level pseudo-code of the model is written. (photo from the paper)

Figure 2 sheds light on the high-level explanations of the previous paragraph with one concrete example. In array A we have n sorted numbers on which our three-level tree performs indexing by a user-defined maximum error of ϵ. As you know for expressing one line we only need its slope and intercept (indicated by S_i^j and ic_i^j in each cell). We also store the first key in each segment to aid searching speed and construction.

The PGM model has a flexible design and can: (1) adjust itself to the query operations distribution, (2) consider potential repetitiveness at the different levels of the tree to make further save space complexity, and (3) auto-tune itself within seconds over nearly billions of keys to satisfy space-time trade-off. It also allows us to have both static and dynamic settings. Other sub-variations are mentioned throughout the paper. I only focus on one of these many variations, due to time constraints. Implementation of the other variations is contingent on how much time the main version will take from me and the workload out of this course (my Data mining course, and defending my thesis proposal due to the end of this semester).

1.3 Base lines

The authors compare their results with B^+-tree and recently learned-index structures, namely RMI, or CSS-tree. I will choose one baseline, most likely the former, and implement it by using the inspiration of existing versions.

2 Tentative Evaluation Strategies

I consider two different criteria, time and space. The time spent executing several queries can simply be measured by the std::chrono library in C++ or similar libraries. Measuring space might become challenging as it is not mentioned in the PGM-indes paper how to accurately measure the space complexity of their model.

3 Test Data Set

The authors use three standard data sets (Web Logs, Longitude, and IoT) each of which has over millions of records. I, however, randomly generate several hundreds of records and store them sorted in a file to test B^+-tree and PGM-index models with that.

4 References

Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *PVLDB*, 13(8):1162–1175, 2020. ISSN 2150-8097. doi: 10.14778/3389133.3389135.