Approximate Pattern Matching in Sparse Multidimensional Arrays Using Machine Learning Based Methods

Anna Kucerova, supervisor Ing. Lubos Krcal
Faculty of Information Technology, Czech Technical University in Prague

Introduction

Sparse multidimensional arrays are a common data structure for effective storage, analysis and visualization of scientific datasets. Approximate pattern matching and processing is essential in many scientific domains.

Previous algorithms focused on deterministic filtering and aggregate matching using synopsis style of indexing. Example of these algorithms is the work of G. Navarro and R. Baeza-Yates, who theoretically designed some of the comparing algorithms used in this thesis, which were improved with the help of Locality Sensitive Hashing.

Solution of this work uses binary format and can be used in a commercial array database.

Solution (implemented algorithms)

Exact Pattern Matching
Brute Force - Systematically finds all exact matches in the whole dataset.
Navaro Baeza-Yates - Splits one dimension of data based on the size of the pattern.

Approximate Pattern Matching
(1) **Approximate Brute Force** - Systematically finds all exact matches in the whole dataset with the possibility to specify the allowed error.
(2) **Fast Filter** - Splits pattern across every dimension and searches for sub-patterns. Potential solutions are checked using dynamic programming.
(3) **Stricter Filter** - Identical to Fast Filter with added fast preverification before dynamic check.
(4) **SimHashed Stricter Filter** - Identical to Stricter Filter but hashes the data before comparison using SimHash.
(5) **LSB Hash Stricter Filter** - Identical to SimHashed Stricter Filter but uses LSB (Least Significant Bit) Hash instead.

Evaluation

Parameters
database size 256 MB
density of data 50%
number of dimensions in the dataset 3
pattern occurences 0,1%

The *find phase* (see Figure 1) is shorter when using hashing, because of faster comparison and reusability of hashes (Figure 2), but on the other hand the overall performance is worse than without hashing (Figure 3), effect caused by a large number of collisions. This problem may be addressed by optimizing the hashing method.

All tested properties:

Database size (256 MB, 4 GB)
Density of the data (2%, 50%, 100%)
Number of dimensions in the dataset (2, 3, 4)
Number of pattern occurrences (0,01%; 0,1%*; 1%)

The tests were performed on integer hypercubic data.

Figure 2: Duration of the *find phase*

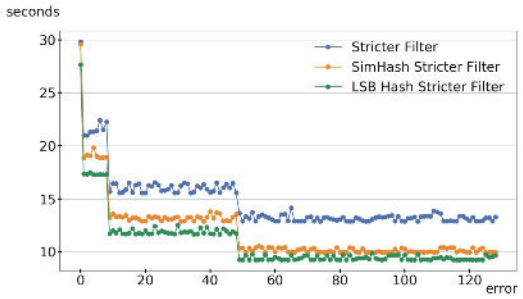


Figure 3: Duration of the whole process

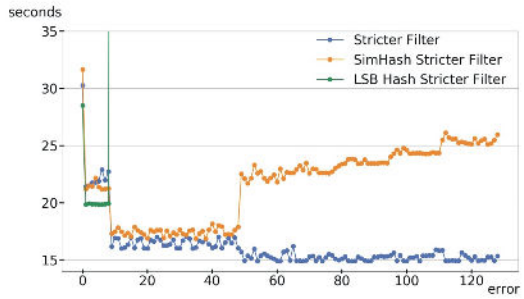
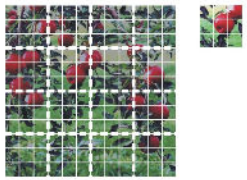
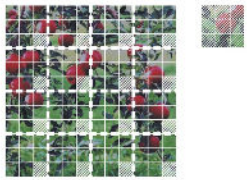


Figure 1: Filtering process of the implemented solution

Initialization
Reads headers of both pattern and data and checks if they match. (1, 2, 3, 4, 5)



Split
Splits pattern into predefined parts and creates a virtual grid over the data with the same proportions. (2, 3, 4, 5)



Hash
Hashes all parts of the pattern and parts of the data used in comparison in the next step. (4, 5)

Find
Finds solutions systematically using all parts of the data. (1)
or
Finds potential candidates using all parts of pattern, but only "bottom-right" corners of the split data. (2, 3, 4, 5)

Preverification
Verifies potential candidates using simple linear comparison. (3, 4, 5)

Dynamic Check
Uses dynamic programming alignment to finalize found solutions. (2, 3, 4, 5)

Baeza-Yates, R.; Navarro, G. New models and algorithms for multidimensional approximate pattern matching. J. Discret. Algorithms, volume 1, no. 1, 2000: pp. 21–49.