BLINKS

A ranked keyword based search algorithm

Diego Omiciuolo Iacopo Mandatelli Mattia Biasin Luca Buriola

University of Padua
25 Gennaio 2017

Summary/presentation structure

1. Keyword Based Search
2. BLINKS
3. Problems
4. Test and Results
5. Conclusions and future works

1. Keyword Based Search

2. BLINKS
3. Problems
4. Test and Results
5. Conclusion and future works

Objectives of keyword based search algorithms

Keyword-based access to structured data aims at letting users express their queries through keywords without prior knowledge of both complex query languages and the underlying structure of data.

- It addresses both intensional and extensional problems in searching and accessing the data.

State of the Art: two major ways

- **Schema based approaches**: exploit the schema information to formulate Structured Query Language queries determined starting from the user keyword queries.
- **Graph based approaches**: relational databases are modelled as graphs, where nodes are tuples, edges are primary-foreign key relationships and the query results are based on the computation of specific structures over the graphs.

Graph-based approach details

The information stored in a database can be captured by a directed graph $G_D = (V, E)$. Each tuple t_v is modeled as a node $v \in V$ in G_D, associated with keywords contained in the corresponding tuple.
For any two nodes $u, v \in V$, there is a directed edge $u \rightarrow v$ if and only if there exists a foreign key on tuple t_u that refers to the primary key in tuple t_v.

Problem definition: query and results

A keyword search query q consists of a list of query keywords $(w_1,..., w_m)$. Given a query q and a directed graph G, an answer to q is a pair $\langle r,(n_1,..., n_m)\rangle$ where r and n_i are nodes in G satisfying:

- For every i, the node n_i contains the keyword w_i.
- For every i, there exists a directed path in G from r to n_i.

1. Keyword Based Search

2. BLINKS

3. Problems

4. Test and Results

5. Conclusion and future works

Blinks overview

- **Search strategy**: BLINKS is based on cost-balanced expansion;
  - Cost-balanced expansion: the algorithm explores the graph starting from nodes containing query keywords and attempts to balance the number of accessed nodes for expanding each cluster. The cluster to expand next is the cluster with the smallest cardinality.
- **Combining indexing with search**: BLINKS augments search with an index, which selectively precomputes and materializes some shortest-path information. This index significantly reduces the runtime cost of implementing the optimal backward search strategy.
- **Partitioning-based indexing**: BLINKS partitions a data graph into multiple blocks. The bi-level index stores summary information at the block level to initiate and guide search among blocks, and more detailed information for each block to accelerate search within the block

Graph partitioning and portals

The partitioning of a graph into blocks can either be done via edge separators or node separators.

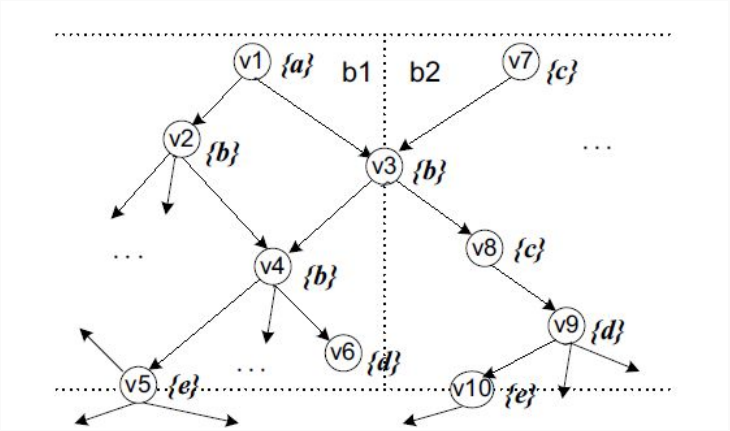
1. We first partitioned the graph using a breadth-first search, obtaining an edge-based partitioning.
2. We transformed this partitioning into a node-based one, using a simple policy to shift the block boundary from the edge to one of the corresponding nodes.

Portals

In a node-based partitioning a node is called a *portal* if it belongs to two different blocks b_i, b_j with $i \neq j$. A portal can be either an *in-portal*, an *out-portal* or both.

- **In-portal**: it has at least one incoming edge from another block and at least one outgoing edge in this block;
- **Out-portal**: it has at least one outgoing edge to another block and at least one incoming edge from this block.

- v_5 is an out-portal for block b_1
- v_3 is both in-portal and out-portal for blocks b_1 and b_2



Intra block index

- **Intra block keyword-node lists**: for each keyword w, $L_{KN}(b,w)$ denotes the list of nodes in b that can reach the node that contain w without leaving b, sorted according to their shortest distances to any node in b containing w.
- **Intra-block node-keyword map**: looking up a node $u \in b$ together with a keyword w in this map returns $M_{NK}(b,u,w)$, the shortest distance from u to w within b.
- **Intra-block portal-node lists**: for each out-portal p of b, $L_{PN}(b,p)$ denotes the list of node in b that can reach p without leaving b, sorted according to shortest distances to p.
- **Intra-block node-portal distance map**: looking up a node $u \in b$ in this hash map returns $D_{NP}(b,u)$, the shortest distance in b from a node u to the closest out-portal of b.

Block index

- **Keyword-block lists**: for each keyword w, $L_{KB}(w)$ denotes the list of blocks containing keyword w, *i.e.*, at least one node in the block is labeled with w.
- **Portal-block lists**: for each portal p, $L_{PB}(p)$ denotes the list of blocks with p as an out-portal.

Implementing Backward/Forward Search Strategy

- **Implementing Backward Search Strategy**: The cost-balanced expansion is implemented by the function *pickKeyword*, which selects the keyword with the least number of explored nodes.
- **Implementing Forward Search Strategy**: we consult the *intra-block node-portal distance map* (D_{np}) for the shortest distance from the node to any out-portal in its block. If the distance turns out to be longer than the intra-block distance from the node to the keyword we conclude that the shortest path between the node and the keyword indeed lies within the block.

Implementation details (1)

- To support backward expansion in multiple blocks we use a *queue* Q_i of *cursors* for each query keyword w_i.
- For each keyword w_i we use the *keyword block-list* (L_{KB}) to find blocks containing w_i.
- A *cursor* is used to scan each intra-block keyword-node list (L_{kn}) for w_i .
- When we reach an in-portal *u* of current block, we need to continue backward expansion in all blocks that have *u* as their out-portal. We can easily identify such blocks by the *portal-blocks list* (L_{pb}).

Implementation details (2)

- For each such block *b*, we continue expansion from *u* using a new cursor, this time to go over the *portal-node list* (L_{pn}) in block *b* for out-portal *u*.
- It is possible for searchBlinks to encounter the same portal node *u* multiple times:
  - *u* can be reached by nodes containing the same keyword in different blocks.
  - *u* can be reached by nodes containing different keywords.
- We use a bitmap *crossed* to keeping track of whether *u* has ever been crossed starting from a query keyword.

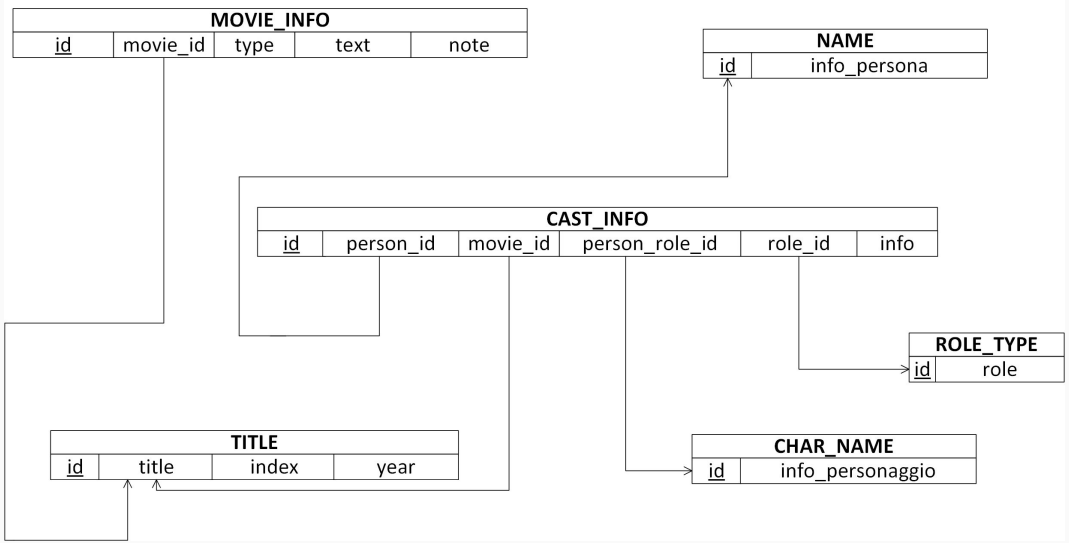
1. Keyword Based Search
2. BLINKS

3. Problems

4. Test and Results
5. Conclusion and future works

IMDB problems

IMDB relational schema:



Tables have only **In** or **Out** references but not both in the same table.

CONSEQUENCE: Out-portal nodes does not exist.

IMDB problems

Some problems rise with the IMDB schema structure:

- The maps L_{pb}, L_{pn} and D_{np} are empty.
- BLINKS search algorithm is useless to apply on this database because it doesn't exploit the advantages of indexing and block division.

In the BLINKS paper, the authors resolve the problem arbitrarily changing the graph structure, using movie titles as nodes and links between movies as edges.
Coffman does not report how he has overcome this problem.

BLINKS memory issue

The overall size of the indexes scales as: $O(\sum_b N_b^2 + BP)$

where N_b is the size of blocks, P is the number of portals and B is the number of blocks.

To overcome the explosion of the needed memory, we implemented the indexes as HashMaps, where the keys are strings with the following format:

- Nodes, Portals: “tableNameprimaryKey”, eg. “desertAtakama”
- Blocks: unique block number
- Keywords: “keyword”, eg. “Sardegna”
- Combinations: “block.node.keyword”, eg. “3.desertAtakama.Sardegna”

If the HashMaps should contain a limit value, for eg. ∞, we don’t put the relative key-value in them, thus saving a lot of memory.

1. Keyword Based Search
2. BLINKS
3. Problems

4. Test and Results

5. Conclusion and future works

Our Experimental Setup

The machine we based the tests on have the following characteristics:

- OS: Windows 10 Home (Version 1607)
- Processor: Intel Core i5-2450M @ 2.50GHz
- Ram: 8 Gb 1600 MHz DDR3
- Memory: SSD SanDisk Plus 240 Gb
- DBMS: PostgreSQL
- Java: Version 1.8.0_111

We limited the maximum JVM allocation memory to 5 Gb.
The following results are referred to the Mondial database, we haven't done the test on IMDB for the previously told reasons.

Experimental measures

- Execution time
- Recall:

$$Recall = \frac{|RelevantDocuments \cap RetrivedDocuments|}{|RelevantDocuments|}$$

- Precision @n:

$$Prec(n) = \frac{1}{n} \sum_{m=1}^{n} a_m$$

  - a_m is the relevance of document d_m.
- Average precision (AP):

$$AP = \frac{\sum_{k=1}^{n} (Precision(k) \times rel(k))}{NumberOfRelevantDocuments}$$

Mondial & Query Numerical Results

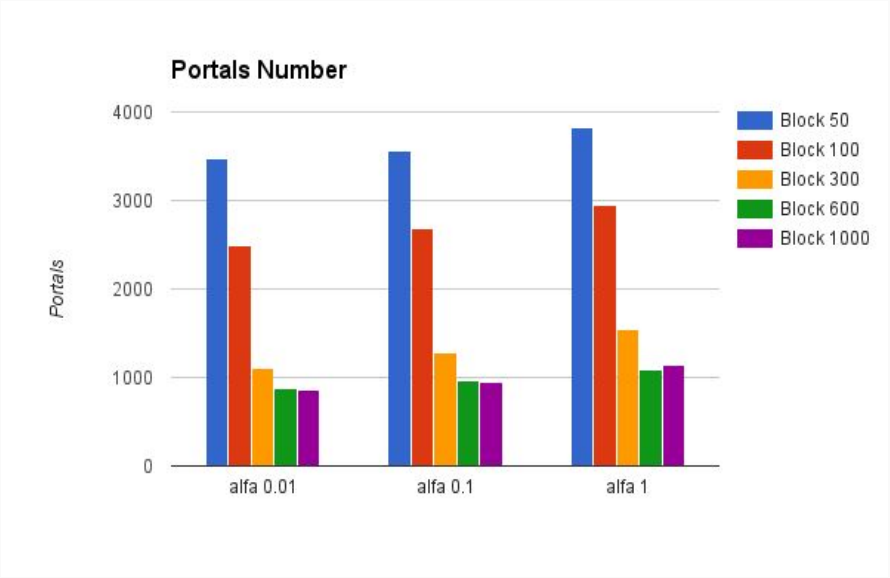
- MONDIAL

Number of tables	Number of tuples
28	20126

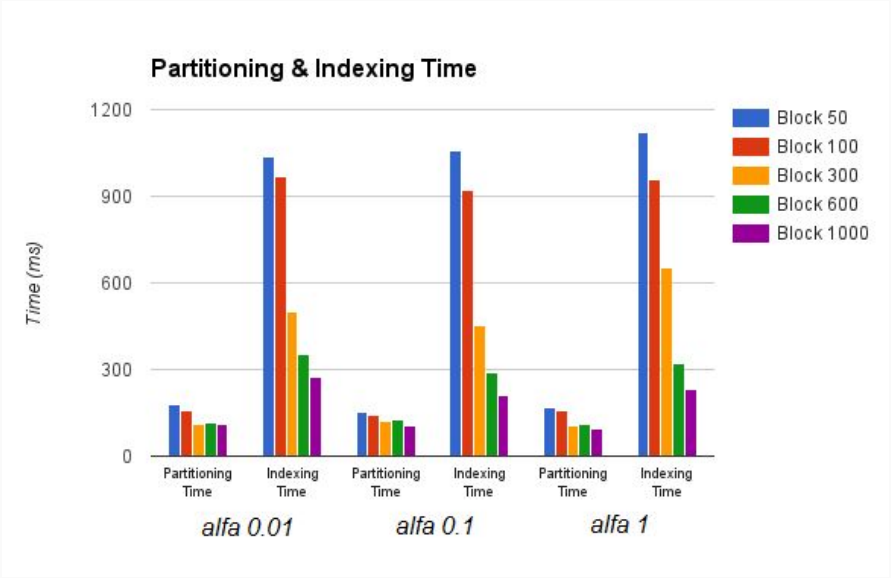
- Query numerical results

Batabase	Completed	Correct	Wrong
Mondial	50	31	19

Portals & Partitioning/Indexing Time

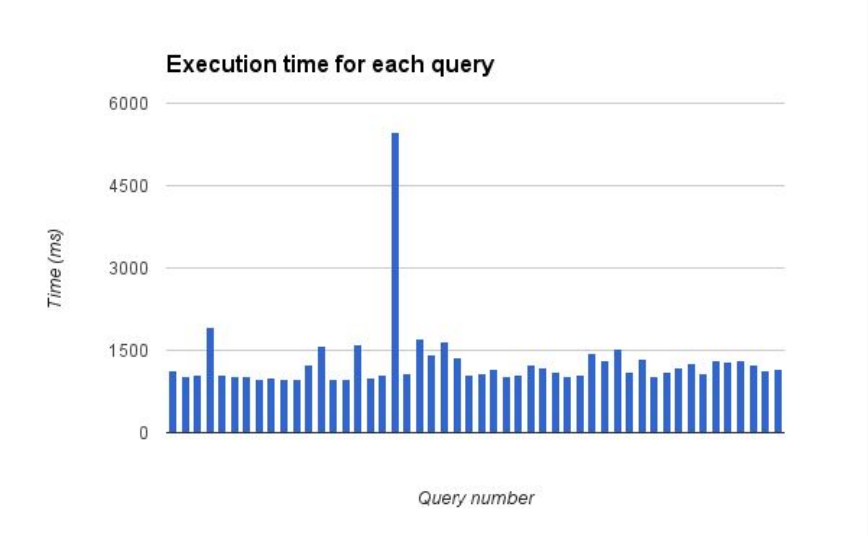


a) *Number of portals for each block dimension varying alfa*

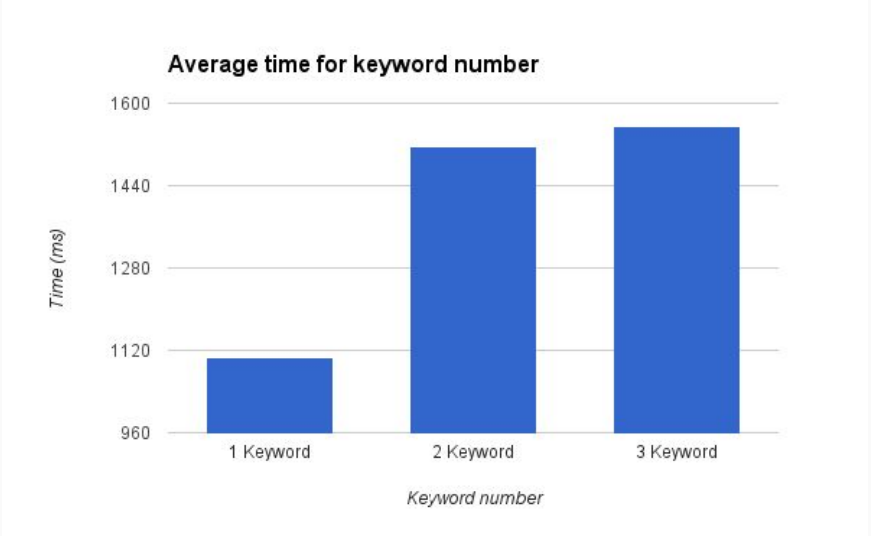


a) *Partitioning and indexing time for each block dimension varying alfa*

Execution times

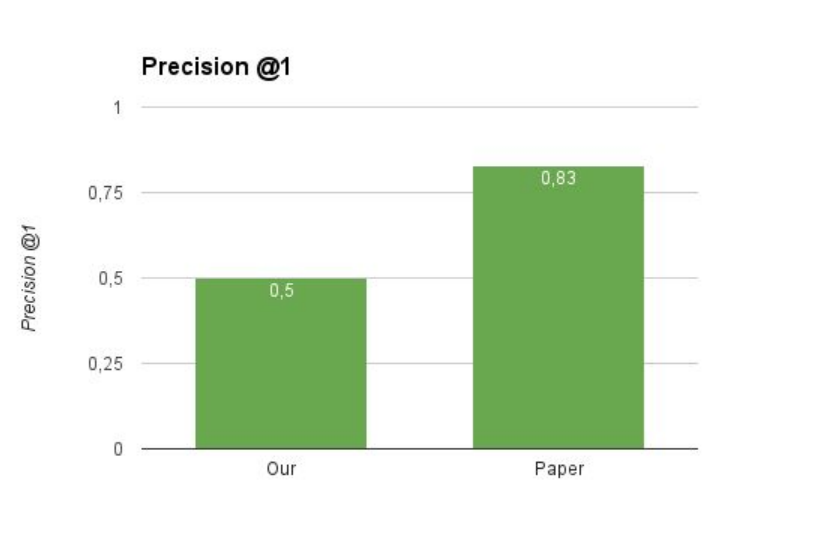


a) *Execution time for each query.*

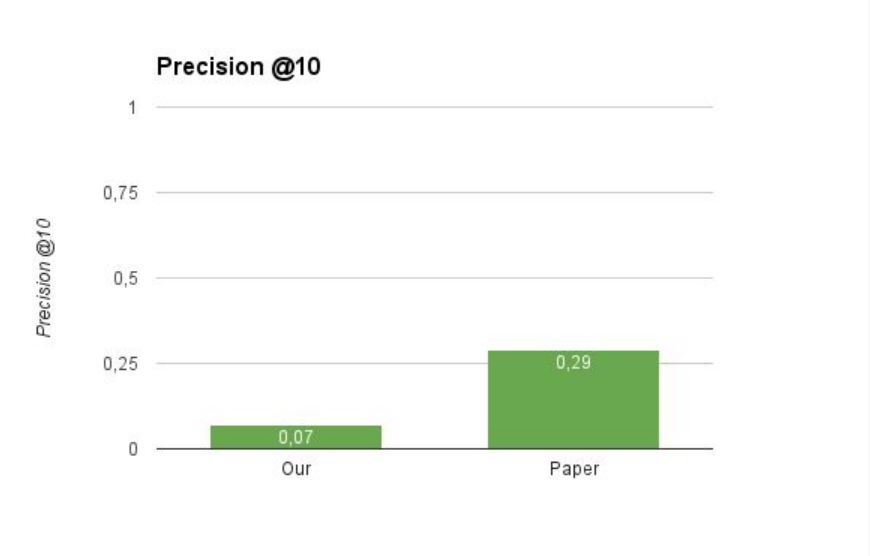


b) *Average execution time for keyword number.*

Average precision

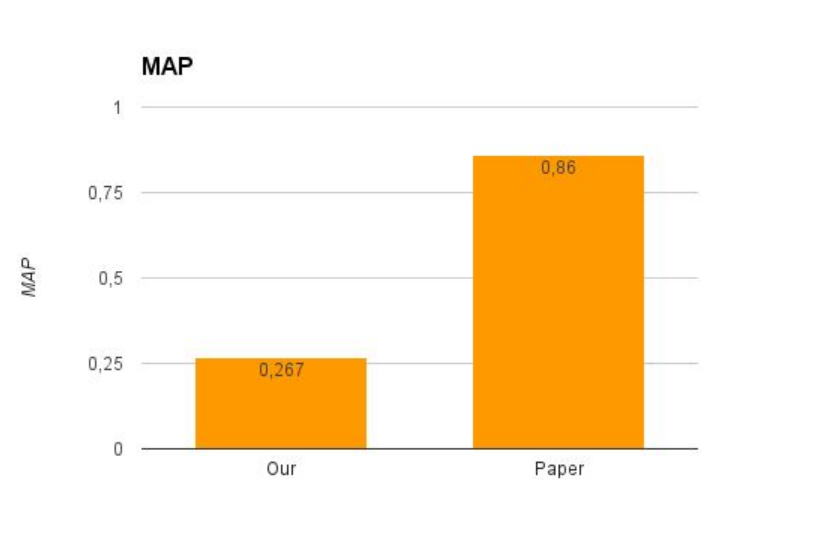


a) *Comparison between our precision @1 and the paper one.*

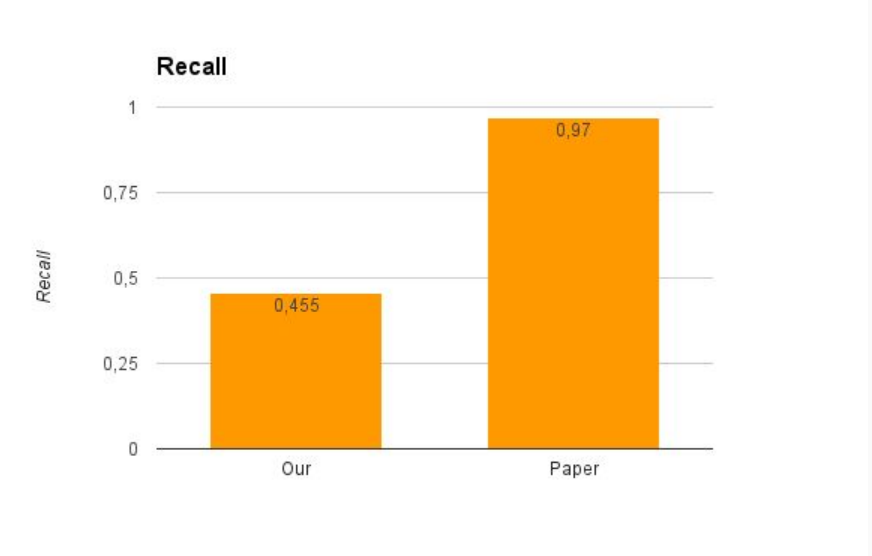


b) *Comparison between our precision @10 and the paper one.*

Map & Recall



a) *Comparison between our MAP and the paper one.*



b) *Comparison between our Recall and the paper one.*

1. Keyword Based Search
2. BLINKS
3. Problems
4. Test and Results
5. Conclusions

Encountered problems

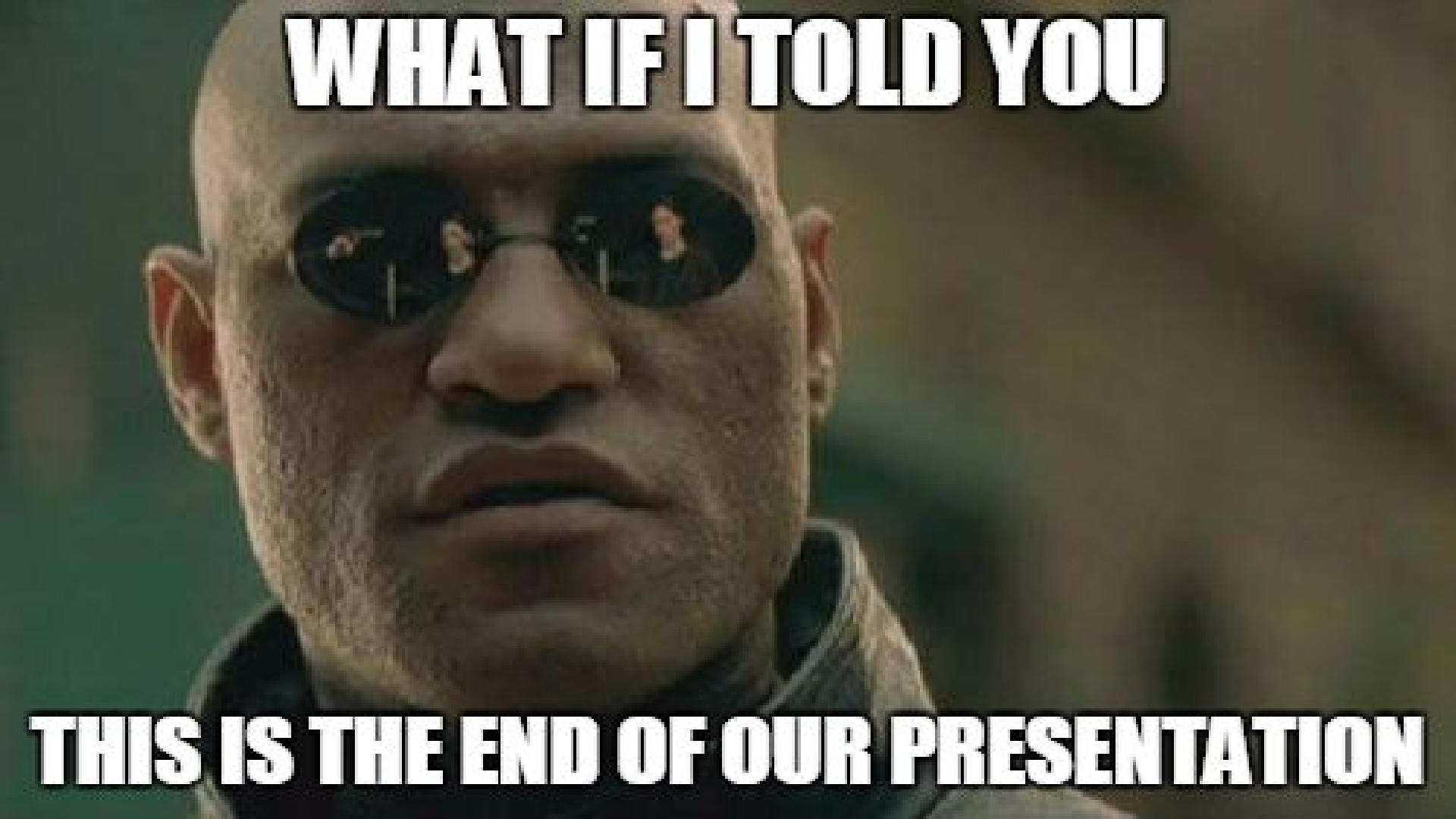
A direct comparison between our results and the ones reported by the authors of BLINKS is a bit difficult for the following reasons:

- They have completely changed the structure of the graph obtained from the IMDB relational schema, we suspect for a better fitting of their algorithm.
- They don’t report any test about queries precision, MAP and recall but only the average execution time of all the queries.
- The BLINKS paper lack some important implementation details and the value of some constants.

Conclusions

From our tests we observed:

- Good time performances due to the abundance of indexing structures that help in the searching start moments.
- A large memory consumption due to the indexing and partitioning structures, as highlighted in the Coffman’s paper.
- Good results with the most simple queries but poor results with the most complex ones.


WHAT IF I TOLD YOU
THIS IS THE END OF OUR PRESENTATION

References

- Java Graph Libraries: http://jgrapht.org/
- H. He, H. Wang, J. Yang, P. S. Yu, “BLINKS: Ranked Keyword Searches on Graphs”, Proceedings of the 2007 ACM SIGMOD international conference on Management of data, June 2007, China.
- J. Coffman, A. C. Weaver, “An Empirical Performance Evaluation of Relational Keyword Search Techniques”, IEEE Transactions on Knowledge and Data Engineering, January 2014, USA.