# Faster BlockMax WAND with Longer Skipping


Antonio Mallia
*New York University*

Elia Porciani
*Sease Ltd.*


ECIR2019 Cologne

## Abstract


**BlockMax WAND** and **Variable BlockMax WAND** represent the most advanced query processing algorithms that make use of dynamic pruning techniques, which allow them to retrieve the top k most relevant documents for a given query without any effectiveness degradation of its ranking.

In this paper, we describe a new technique for the BlockMax WAND family of query processing algorithm, which improves block skipping in order to increase its efficiency. We show that our optimization is able to improve query processing speed on short queries by up to 37% with negligible additional space overhead.


## Contribution

- We propose an optimization for the BlockMax WAND (BMW) family of algorithms, which exploits particular sequences of block max scores in order to perform longer skipping.
- We embed an additional data structure that stores precomputed skips in order to overcome the run time search overhead introduced by compressing the block boundaries.

**github.com/pisa-engine/pisa/tree/ecir19-ls**

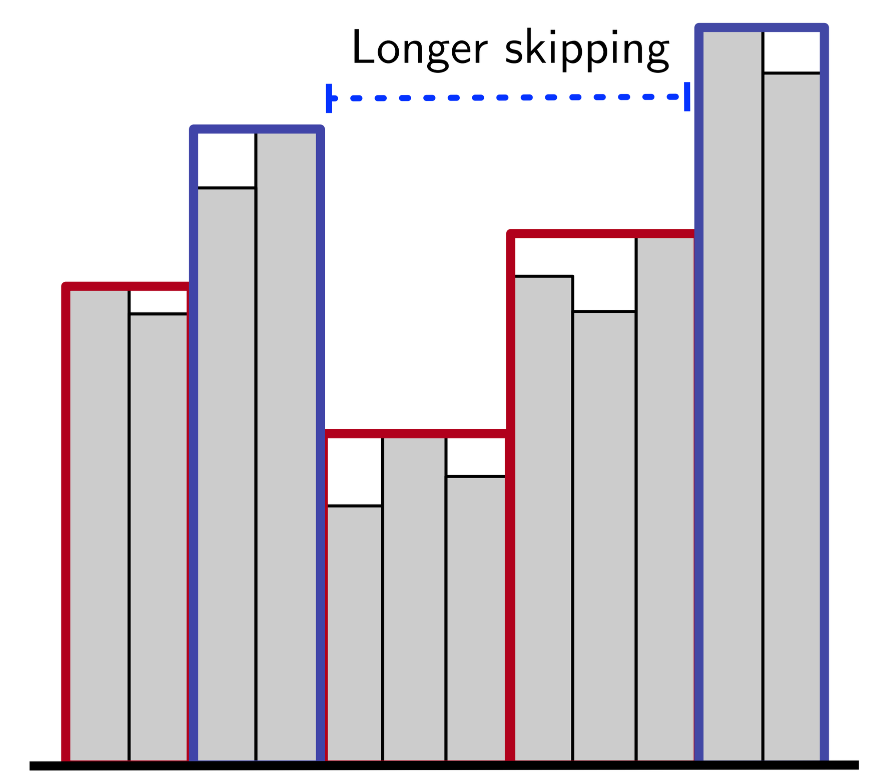
## Longer Skipping

**Longer Skipping (LS)** is a new strategy to advance the term iterator farther than the current block boundaries, identifying the next document ID to move to by progressively skipping entire blocks until one with greater block-max score is found.

**Precomputed Longer Skipping (PLS)** precomputes the skip size at index build time, encodes it with a fixed number of bits and stores it interleaved with the blocks information.

**Algorithm 1** Find next doc ID

```
next_docid ← MAX_DOCID
for all t ∈ terms do
    block ← t.block
    s ← block.score
    docid ← block.boundary
    while block.score <= s do
        docid ← block.boundary
        block ← block.next
    end while
    if docid < next_docid then
        next_docid ← docid
    end if
end for
```

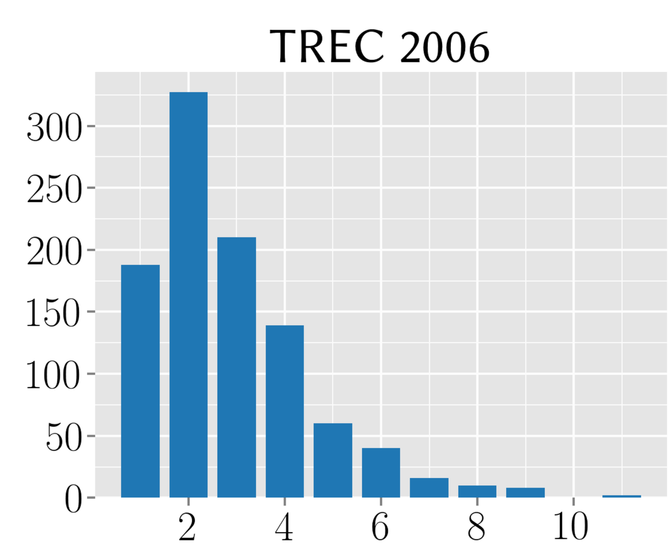


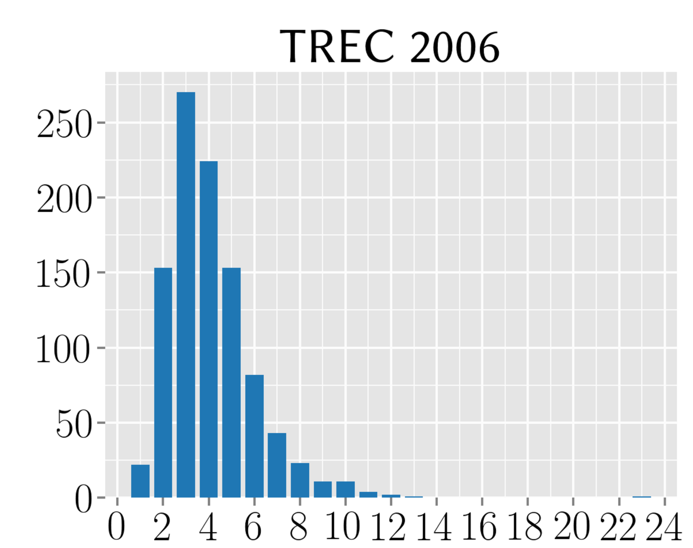

## Experiments

*Datasets: we* performed our experiments on standard datasets: **Gov2** and **ClueWeb09 Category B**.

| | Gov2 | ClueWeb09 |
|---|---|---|
| Documents | 24,622,347 | 50,131,015 |
| Terms | 35,636,425 | 92,094,694 |
| Postings | 5,742,630,292 | 15,857,983,641 |

*Queries*: to evaluate the speed of query processing we use the **TREC 2005** and **TREC 2006 Terabyte Track Efficiency Task**




*Baseline*: BMW and VBMW with 40 elements per block in average, in both their uncompressed and compressed form.

Improvements for short queries (from 2 to 4 terms) with negligible performance degradation for longer queries.

For the compressed version of the algorithms the run time optimization does not lead to any improvements. The precomputed version overcomes this issue and obtains almost the same gain of the run time version for the uncompressed BMW with a negligible overhead in index size.

| | | Gov2 | | | | | ClueWeb09 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | 3 | 4 | 5 | 6+ | 2 | 3 | 4 | 5 | 6+ |
| TREC 2005 | BMW | 1.22 | 3.07 | 4.68 | 7.43 | 16.73 | 4.63 | 11.37 | 16.68 | 25.72 | 55.99 |
| | VBMW | 0.99 | 1.91 | 2.69 | 4.21 | **9.18** | 3.17 | 6.39 | 8.92 | 14.46 | 32.04 |
| | BMW-LS | 0.93 | 2.88 | 4.61 | 7.41 | 17.40 | 2.92 | 10.20 | 16.76 | 26.84 | 60.42 |
| | VBMW-LS | **0.78** | **1.77** | **2.63** | **4.20** | 9.23 | **2.18** | **5.66** | **8.57** | **14.44** | **31.95** |
| | C-BMW | 1.33 | 3.39 | 5.13 | 8.27 | 18.26 | 5.19 | 12.78 | 19.09 | 29.19 | 63.32 |
| | C-VBMW | 1.10 | 2.08 | 2.93 | **4.60** | **10.16** | 3.53 | 6.97 | 9.86 | **16.06** | 36.26 |
| | C-BMW-LS | 1.38 | 3.42 | 5.32 | 8.26 | 18.74 | 5.36 | 13.11 | 19.42 | 29.93 | 65.08 |
| | C-VBMW-LS | 1.14 | 2.15 | 3.04 | 4.75 | 10.21 | 3.67 | 7.34 | 10.29 | 16.46 | 36.48 |
| | C-BMW-PLS | 1.12 | 3.10 | 5.00 | 8.00 | 18.77 | 3.89 | 11.19 | 18.41 | 29.58 | 65.80 |
| | C-VBMW-PLS | **0.94** | **1.95** | **2.93** | 4.71 | 10.17 | **2.68** | **6.30** | **9.52** | 16.07 | **36.01** |
| TREC 2006 | BMW | 1.11 | 3.58 | 6.24 | 10.03 | 23.85 | 3.46 | 11.33 | 19.82 | 32.37 | 74.13 |
| | VBMW | 0.78 | 2.26 | 3.58 | 5.55 | **12.88** | 2.28 | 6.80 | 11.64 | 18.68 | 42.17 |
| | BMW-LS | 0.85 | 3.22 | 6.08 | 9.98 | 24.86 | 2.50 | 10.42 | 19.77 | 33.28 | 80.62 |
| | VBMW-LS | **0.58** | **2.05** | **3.46** | **5.49** | 12.92 | **1.66** | **6.25** | **11.35** | **18.59** | **42.04** |
| | C-BMW | 1.22 | 3.90 | 6.95 | 11.09 | 26.34 | 3.80 | 12.48 | 22.27 | 35.83 | 82.96 |
| | C-VBMW | 0.89 | 2.49 | 3.96 | 6.08 | 14.60 | 2.51 | 7.42 | 12.86 | **20.40** | **46.87** |
| | C-BMW-LS | 1.28 | 4.02 | 7.19 | 11.17 | 27.07 | 3.96 | 12.91 | 22.85 | 37.02 | 85.59 |
| | C-VBMW-LS | 0.91 | 2.57 | 4.06 | 6.23 | 14.88 | 2.61 | 7.68 | 13.21 | 20.99 | 47.48 |
| | C-BMW-PLS | 1.02 | 3.57 | 6.56 | 10.75 | 26.52 | 3.09 | 11.57 | 21.92 | 36.52 | 85.45 |
| | C-VBMW-PLS | **0.72** | **2.34** | **3.86** | **6.07** | **14.54** | **1.98** | **6.88** | **12.51** | 20.46 | 47.33 |

## Conclusions and Future Work

Applied a longer skipping strategy to both BMW and VBMW, which results in marked benefits of processing time for short queries.

We intend to study how beneficial can be the combination of our longer skipping strategy to threshold estimation techniques.


**Acknowledgments**. Antonio Mallia's research was partially supported by NSF Grant IIS-1718680 "Index Sharding and Query Routing in Distributed Search Engines".