# PdF Project: Space-Efficient Encodings for Compressed Self-Indices

Advisor: Florian Kurpicz

## Background

There is an ever-increasing amount of textual data created, e.g., due to the exponentially increasing capability to sequence genetic data. Furthermore, there are code repositories such as GitHub and natural text collections such as Wikipedia. One common property of these types of input is that they are highly repetitive (humans share 99.9 % of their DNA and different versions of source code/Wikipedia articles usually differ only minimally). When processing this type of textual data, we want to use text indices that allow us to speed up queries.

In this project, we mainly focus on three queries: (1) An *access* query allows us to retrieve any character of the input, (2) the *rank* of a character is the number of occurrences of that character before a given position, and (3) a *select* query returns the position where a character with a given rank occurs.

For these queries, the *wavelet tree* [2] is currently the de-facto standard. A wavelet tree can also be compressed—using entropy coding—reducing the required space of the index. However, this type of compression is blind to repetitions. The *block tree* [1], however, can also be used to answer these queries and is based on the Lempel-Ziv compression [4], which results in a very good compression ratio for highly repetitive inputs. While the block tree can be used as a replacement for wavelet trees, there are still open problems. First, block trees are very slow to construct (compared to wavelet trees). While there are new, significantly faster construction algorithms, they also require a much more working space [3]. Second, on non-highly repetitive inputs, their size is comparable to wavelet trees, they are slower to construct, and they only achieve a similar query performance.

## Objective

Currently, block trees are only significantly more space-efficient than wavelet trees for highly repetitive inputs, where they require less than 1 bit per symbol, even with rank and select support. In this project, we take a closer look at efficient encodings for block trees to also make them more space-efficient than wavelet trees on non-highly repetitive inputs. To this end, we will utilize techniques used in Lempel-Ziv compression, e.g., considering only occurrences in a sliding window to reduce the size of backward pointers and a compressed representation of plain characters in the block tree's leaves.

A possible extension of this project is the parallel (shared or distributed memory) construction of the encoded block trees. Here, we might discover new scalable construction algorithms, as some techniques (like the use of a sliding window) remove dependencies that occur in a (canonical) block tree.

## References

[1] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *J. Comput. Syst. Sci.*, 117:1–22, 2021.

[2] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850. ACM/SIAM, 2003.

[3] Dominik Köppl, Florian Kurpicz, and Daniel Meyer. Faster block tree construction. In *ESA*, volume 274 of *LIPIcs*, pages 74:1–74:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[4] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.