CSCI 6057 Advanced Data Structures – Project Proposal

Yansong Li
B00755354

1. Introduction.

Due to the broad range of applications in bioinformatics, indexing highly compressible strings for rapid queries has become a prominent research area in string processing. Although a large number of indexes for extremely competitive strings have been presented thus far, developing compact indexing that enables varied queries such as rank and select is still a difficulty [1]. Burrows-Wheeler Transform (BWT) is a lossless data compression tool by a reversible permutation of an input string, and run-length compressed BWT (RLBWT) is BWT with run-length encoding [2]. L-F mapping is the key function of BWT and RLBWT for string reversion, and the current best result is from Nishimoto et al. who achieve to compute it in $O(1)$ time and $O(r)$ (r is the number of runs in RLBWT) space using table lookups in their OptBWTR structure [1]. However, due to the enormous size of datasets in the field of bioinformatics such as human chromosomes, the O(r) space look-up table would still need to be stored on memory, and the number of cache misses would be large because of the randomness of the jumps of L-F mapping.

Is there an approach to arrange the rows in the look-up table of RLBWT to reduce the number of cache misses and provide a better memory locality? Sirén et al. described a method in their Graph BWT (GBWT) paper for achieving a better memory locality of the graph extension of the positional BWT (gPBWT) [3]. The method describes that we divide the BWT into sub-BWTs according to the most significant character in the lexicographic ordering, rather than storing it as a single string on the desk; also, the sub-BWTs are used as blocks for rank queries over the BWT with proper assumptions [3]. Hence, the project is trying to see if we apply Sirén et al.'s method to the look-up table of RLBWT from Nishimoto et al. will lead to a lower number of cache misses while maintaining the $O(1)$ time of L-F mapping. If it does, can we have similar assumptions as Sirén et al had? If both questions' answer is yes, then we can develop a cache-friendly RLBWT.

2. Background Description

(i) Nishimoto et al.'s novel data structure

In order to store an arbitrary permutation π on string $S = \{0, \dots, n-1\}$ with a property that the sequence $\pi(0), \pi(1), \pi(2), \dots, \pi(n-1)$ is composed of a limited number b of unbroken incrementing subsequences, $O(b)$ space compressed spare bitvector was widely used [2]. Nishimoto et al. proposed an alternative $O(b)$ space data structure that stores a sorted table of π. There is a quadruple for each subsequence head p, which consists of the head of the subsequence p; the length of the subsequence $|p|$; the sequence $\pi(p)$; and the index of the subsequence containing $\pi(p)$ [2].

(ii) Sirén et al.'s method

GBWT is an FM-index of multiple texts over an integer alphabet V, for each vertex $v \in V$, a local alphabet is defined as $\sum v = \{w \in V | (v, w) \in E)$. \$ is added to $\sum v$ if v is last vertex

of a path, and $\sum\$$ is defined as the initial vertex of each path [3]. A BWT is partitioned to substrings BWT_v corresponding to the prefixes end with v, and each BWT_v is encoded using $\sum v$. Let $\sum v\,(w) = k$ when $w \in \sum v$ is the k^{th} character in $\sum v$ in sorted order [3]. Meanwhile, a cache-friendly graph layout is assumed, which means $v \in V$ have low out-degree, and the graph is almost linear and almost topologically sorted [3]. Next, the BWT_v will be rearranged using GBWT encoding [3].

3. Project Plan

The experiment will be conducted based on look-up tables of RLBWT from Brown et al., who actually implemented the quadruple table that was described in Nishimoto et al.'s paper [2]. Next, we are going to cut the RLBWT table into blocks, and the size of blocks will be determined by the cache size (L3 cache for now). Then, blocks will be used as vertices $v \in V$ and jumps of L-F mapping that across different blocks will be used as edges $e \in E$ to build a graph $G = (V, E)$. Each e in G will be weighted and the weight $w \in W$ of e will be determined by the total length of runs between two vertices $(v, u) \in V$. Meanwhile, G will be undirected since each w is the sum of runs from both directions of the edge.

After we have generated the graph using the RLBWT table, we will cluster it using METIS, and the sizes of the cluster will also be determined by the cache size. The clustering algorithm is finding $e = (v, u)$ with the largest w then we put v and u into the same cluster and keep clustering e with next the largest w until reaching the size of a cluster. When we have partitioned all the blocks into clusters, we calculate the total weight W_r of inter-clusters edge $e_t \in E_t$. As a comparison, we will also try to partition the graph sequentially with the same cluster size. Finally, we compare the total weight W_s of sequential cluster with W_r. If $W_r < W_s$ significantly, then it shows that Sirén et al.'s method does decrease the number of cache misses of the RLBWT table in Nishimoto et al.'s construction. In addition, if the degree of V is limited and the expansion of G is low (similar assumptions as Sirén et al.'s method), then we could build a cache-friendly RLBWT based on the graph and clustering.

References

[1] Nishimoto, T., & Tabei, Y. (2021). Optimal-Time Queries on BWT-Runs Compressed Indexes. *ICALP.*

[2] Brown, N.K., Gagie, T., & Rossi, M. (2021). RLBWT Tricks. *ArXiv, abs/2112.04271*.

[3] Sirén, J., Garrison, E., Novak, A., Paten, E., & Durbin, R. (2020). Haplotype-aware graph indexes. *Bioinformatics,* Volume 36, Issue 2, pp. 400–407.

Except the papers in the reference, I am also going to read these papers for the project (not the final list):

1. Chien, YF., Hon, WK., & Shah, R. *et al.* (2015). Geometric BWT: Compressed Text Indexing via Sparse Suffixes and Range Searching. *Algorithmica,* 71, pp. 258–278.

2. M¨akinen, V., Navarro, G., Sirén, J., & V¨alim¨aki, N. (2010). Storage and retrieval of highly repetitive sequence collections. *J. Computational Biology*, pp. 281–308.